FBNOC: FPGA BASED NETWORK ON CHIP SIMULATOR

BY

GAMIL ABDULLAH MOHSEN AHMED

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

MAY 2017

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **GAMIL ABDULLAH MOHSEN AHMED** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.

Thesis Committee

Dr. Muhammad Elrabaa (Adviser)

(Co-adviser)

Dr. Aiman El-Maleh (Member)

Dr. Tarek Sheltami (Member)

(Member)

Dr. Ahmad Almulhem
Department Chairman

Dr. Salam A. Zummo
Dean of Graduate Studies

1/6/17
Date

Dedication

To my parents, wife, children, brothers and sister for their endless support and love.

ACKNOWLEDGMENTS

All praise and thanks be to Almighty Allah, the one and only who helps us in every aspect of our lives.

Acknowledgement is due to King Fahd University of Petroleum and Minerals for giving me this precious opportunity to resume my Master degree.

I would like to express deep gratefulness and appreciation to my Thesis advisor Dr. Muhammad El-Rabaa for his continuous help, guidance, and encouragement throughout the course of this work. He spent a lot of his precious time helping me and advising me at each step.

Additionally, I would like also to thank my Thesis committee members: Dr. Aiman El-Maleh and Dr.Tarek Sheltami for their great help and cooperation, which contributed significantly to the improvement of this work.

Finally, my heartfelt gratitude goes to my parents, wife, children , brothers and sister for their encouragement, prayers, and moral support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

NAME: Gamil Abdullah Mohsen Ahmed

TITLE OF STUDY: FBNoC:FPGA Based Network on Chip Simulator

MAJOR FIELD: Computer Engineering

DATE OF DEGREE: May 2017

Due to the increasingly developing technology of silicon fabrication, multi-core systems have been growing rapidly. Future Chip Multi-Processor (CMP) with tens to hundreds of nodes will require an efficient and scalable on-chip communication as traditional bus-based interconnects suffer from lower throughput significantly. Networks-on-Chip (NoC) are being progressively adopted for multi-core inter-communications. According to recent studies, Field Programable Gate Array (FPGA)-based NoC simulators are utilized to study NoC designs. However, these simulators are limited by the available FPGA resources. Hence, existing FPGA-based NoCs consume considerable portion of the FPGA resources. In addition, to simulate different designs, the FPGA-based simulators require a modification in HDL code that in turn requires complete compilation and synthesis of the FPGA design which consumes much time.

The aim of this thesis is to develop a resource-efficient hardware NoC simulator, an FPGA-based NoC (FBNoC) simulator, that is fast, accurate and can be integrated with many-core architectural simulators. In addition, it can be used as a stand-alone NoC simulator with traffic traces or synthetic traffic. The simulator can model and simulate several popular NoC topologies accurately and efficiently without the need to re-synthesize for different NoCs. To do so, the FBNoC includes a multi-variable regression latency model that can calculate the latency per packet for the simulated network accurately and efficiently. The actual NoC topology used in the proposed simulator is a scalable bidirectional ring network (more than one ring) to deliver a packet to its destination, increase the overall system throughput and cooperate with other architectural simulators. It can be integrated with an architectural simulator with up to 256 cores. Moreover, our FBNoC allows trading-off simulation speed for area. In doing so, the simulator uses a multi-local port strategy to reduce the FPGA resources utilization and end-to-end delay. In this work, the design is tested by ChipScope tool. Also, the effect of multi-local port strategy on the FPGA resources is studied. In addition, the performance of the simulated network is examined under synthetic (specific configurations) and realistic traffic. Finally, the average packet latency is compared against the Booksim simulator. The proposed simulator can achieve more than 20000x speedup over the Booksim simulator. Also the FPGA resources utilization are compared with DART and FIST simulators.

ملخص الرسالة

الاسم الكامل: جميل عبد الله محسن أحمد

عنوان الرسالة: محاكي شبكة على الرقاقة مؤسسة على FPGA

التخصص: هندسة حاسوب

تاريخ الدرجة العلمية: 2017

بسبب التطور المتزايد في تكنلوجيا تصنيع السليكون اصبحت الانظمة متعددة الانوية تنموا بشكل سريع. المعالجات المستقبلية متعددة الانوية بعشرات الى مئات الانوية سوف تتطلب اتصال كفوء ومرن حيث ان الاتصالات المرتكزة على الناقل تعاني من تدهور كبير في الانتاجية. الشبكة على الرقاقة (NoC) تتلائم تدريجيا لربط الانظمة متعددة الانوية. بالاشارة الى الدراسات الحديثة فان محاكيات الشبكة على الرقاقة المبنية على مصفوفة بوابة البرمجة الحقلية (FPGA) تستخدم لدراسة خصائص تصميم الـ NoC. ومع ذلك فان هذة المحاكيات محدودة بموارد الـ FPGA المتوفرة. لذلك محاكيات الـ NoC الموجودة تستهلك جزء كبير من موارد الـ FPGA. بالاضافة الى ذلك محاكات تصاميم مختلفة يحتاج الى تحديث في شفرة HDL والذي بدوره يتطلب تاليف وتركيب كامل على تصميم الـ FPGA والذي يستهلك وقت كبير.

الهدف من هذه الرسالة هو تطوير محاكي كفوء الموارد،محاكي قائم على FPGA (FBNoC)، سريع ودقيق ويستطيع التكامل مع محاكيات معمارية متعددة الانوية. اضافة الى ذلك يمكن استخدامة كمحاكي للـ NoC مع ترافك تريس او ترافك اصطناعية. المحاكي يستطيع محاكاة عدة تصاميم بشكل كفوء ودقيق دون الحاجة الى اعادة تركيب وتاليف. لفعل ذلك فان الـ FBNoC يتظمن مودل انحدار تاخير متعدد المتغيرات والذي يستطيع حساب تاخير الحزمة للشبكة التي يحاكيها بشكل كفوء ودقيق. تصميم الشبكة الفعلية المستخدمة

في المحاكي المقترح هو التصميم الحلقي ذو اتجاهين والمرن (اكثر من حلقة) لتوصيل الحزمة الى وجهتها ولرفع الانتاجية الكلية للنظام والتعاون مع محاكيات الهاردوير الاخرى. يمكن ان يدمج مع محاكيات المعمارية حتى 256 نواة. واكثر من ذلك فان الـ FBNoC يعمل موازنة بين المساحة والسرعة. ولفعل ذلك المحاكي يستخدم تقنية المنافذ المحلية المتعددة لتقليل استهلاك موارد الـ FPGA والتاخير من طرف الى طرف. في هذا العمل التصميم يفحص باستخدام اداة ChipScope. ايضا تاثير استخدام استراتيجية المنافذ المحلية المتعددة على موارد الـ FPGA تدرس. بالاضافة الى ذلك اداء الشبكة المحاكاة تختبر تحت تأثير ترافك مصطنعة (اعدادات مخصصة) وترافك حقيقية. معدل التاخير للحزمة يقارن مع المحاكي Booksim. المحاكي المقترح يستطيع ان يحقق سرعة اكثر من 20000 مرة اعلى من الـ Booksim. واخيرا استخدام موارد الـ FPGA تقارن مع محاكي DART و FIST.

CHAPTER 1

INTRODUCTION

1.1 Introduction

MULTI-CORE systems are increasingly used in order to optimize the dedicated applications. With the advancement of technology, the Multi-core systems on-chip (MCSoC) designs become difficult due to increasing number of cores. These systems are becoming increasingly complex in order to fulfill ever more features. To increase the performance and meet the functionality, the systems with more processing elements must be used. To optimize these systems, two main areas may be analyzed: the handling of calculation and communication between the different elements. Different designs have been studied to solve the communication between the different elements. Shared buses have been used for along time. However, they suffer from significantly lower throughput. Future Chip Multiprocessor(CMP) with tens to hundreds number of nodes will require an efficient and a scalable on-chip communication. They are becoming a trend in System on Chip (SoC)

and processor designs. Networks-on-Chip (NoC), first proposed in [1], are being progressively adopted for multi-core inter-communications.

Software simulators are extensively utilized to evaluate the NoCs design trade-offs. Such simulators are also used to simulate the interconnection component within complete systems simulators [2, 3]. Full system simulators, however neither provide accurate simulations nor can simulate a system with a large number of cores in reasonable time. Stand-alone NoC simulators [4] can only study NoC designs independently. They are significantly faster than full-system simulators.

Software simulators are easy to configure, compile, and run. However, for large NoCs, they are very slow and their speed degrades rapidly as the number of cores increases and/or the packet injection rate increases. This drawback can be realized from a performance analysis of Booksim, one of the most popular software simulators [5], the simulator speed can be affected by network size and injection rate. The speed of software-based simulators decreases rapidly as the number of cores and/or injection rate increases. This is because the increasing of injection rate consumes a longer time due to the large number of tasks that need to be done. Fig. 1.1 (a) shows the slowdown in simulation speed of BookSim at an injection rate of 0.01 (packets/node/cycle) when changing the simulated NoCs size from 8 nodes to 169 nodes. The use of thread level parallelism to increase the speed of software-based simulator is very difficult due to the large amount of synchronization in the simulated network. In addition, they suffer from additional synchronization and communication delays when they are coupled with architec-

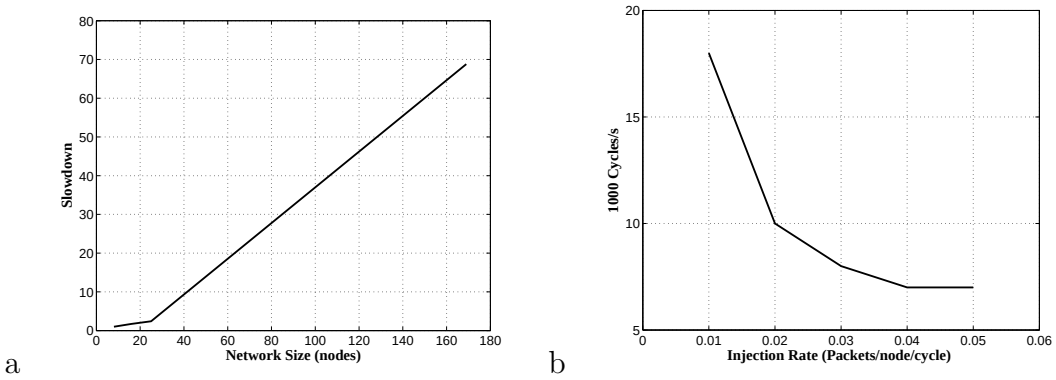


Figure 1.1: (a) Slowdown in simulation speed at an injection rate of 0.01 (packet/node/cycle) when the simulated node size changes from 8 to 169. (b) Simulation speed of Booksim with different injection rates.

tural simulators which slows the whole systems simulation.

The emergence of high-performance, high-density Field Programmable Gate Arrays (FPGA) enabled the emulation of whole systems on a single chip, including NoCs [6, 7]. Full systems emulation however, requires high implementation and verification efforts, and suffers from scalability limitation due to the FPGA capacity constraints. FPGA-based NoC simulators were proposed as an alternative to emulation [8, 9, 10]. These simulators are built by combining all parts of a simulator together on the same chip to exploit all coarse and fine grain parallelisms among simulator events in a NoC. As a result, FPGA-based simulators reduce the simulation time by several orders of magnitude.

1.2 Thesis Motivation

In this section, the main motivations for this work are briefly described which are enumerated as follows:

- Software simulators speed decreases when network size and injection rate increase as discussed before. In addition, the full system simulators suffer from synchronization and communication delay.

- The size of NoC that can be simulated by an FPGA-based simulator is limited by the available FPGA resources, a challenge for large NoC design. Hence, existing FPGA-based NoCs either consume considerable portion of the FPGA resources or suffer from very low frequency. To overcome the limited resources problem, some FPGA-based simulators simplify the router architecture or utilize a virtualization scheme, such as in in DART [10]. The routers area was reduced by decreasing the number of channels and the router pipeline. Another approach to reduce the complexity of the design and FPGA resource utilization is to utilize time-division multiplexing (TDM) [11]. However, additional memory to hold the traffic passing between nodes was added. Thus, the main disadvantages of TDM is the wasted cycles to change cluster states which becomes the dominant time overhead when the ratio of clusters to nodes increases. In FIST [12], each router in the network was modeled as a set of a load-delay curve. Moreover, utilizing multiple FPGAs [13] and off-chip memory [9, 14] can mitigate the limitation of FPGA resources. However, these approaches not only lead to a higher cost but also make the entire system slower and more complex.

- To simulate different topologies, a modification is required in a simulator that requires amendment in HDL code. The modification in code, in turn,

requires a complete compilation and synthesis of FPGA design which consumes much time.

- More importantly, most FPGA-based simulators can not be attached to architectural simulators.

1.3 Thesis Contribution

This section briefly describes the main contributions of this work which are enumerated as follows:

- Developed a resource-efficient hardware NoC simulator that is fast, accurate, and can be integrated with many-core architectural simulators (i.e., it delivers the actual packet with appropriate time stamps). In addition, it can be used as a stand-alone NoC simulator with traffic traces or synthetic traffic. As such, an FPGA-based NoC (FBNoC) simulator is proposed. The simulator can model and simulate several popular NoC topologies accurately and efficiently without the need to re-synthesize for different NoCs

- The multi-local port strategy with ring network is utilized to allow trading off simulation speed for area.

  The actual NoC topology used in the proposed simulator is a scalable bidirectional ring network (more than one ring) to deliver a packet to its destination, increase the overall system throughput, and cooperate with other architectural simulators. However, in a ring network, as the number of nodes

increases, the end to end delay increases and more FPGA resources are consumed. To cope with these problems, we utilized a mapping strategy for multi-local port routers to reduce the FPGA resources utilization and the end to end delay.

- Developed a multi-variable regression latency model that can accurately and efficiently calculate the latency per packet for the simulated network depending on the network size, traffic injection rates, and number of virtual channels per router. Moreover, the latency model enables the simulator to simulate other topologies without re-synthesizing the design.
- The FBNoC can be integrated with an architectural simulator with up to 256 cores.

1.4 Thesis Organization

This thesis is organized as follows: Chapter Two describes the principals of NoC, flow control, routing algorithm, switching technique. It provides the required background to understand the rest of the thesis. This chapter also presents a review of the state of the art in NoC simulations. A description of the proposed FBNoC simulator is presented in Chapter Three. This includes the simulator architecture, its latency model, and the implementation details of its underlying ring network. In Chapter Four, the design is tested by ChipScope tool. Also, the effect of multi-local port strategy on the FPGA resources is studied. In addition, experimental results and comparisons with other NoC simulators are presented

under synthetic and realistic traffic. Finally, Chapter Five concludes the thesis work and highlights some main contributions and results.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Background

In this section, the background of some topics related to our work will be discussed. The concept of NoC will be investigated and the communication structure will be discussed as the main problem in the system interconnection. In addition, the basic component of NoC will also be introduced. Then, the routing, switching, and flow control will be explored. After that, the traffic models will be discussed briefly. In the second part of this chapter, the state of art of simulators NoC will be explored.

2.1.1 Interconnection Structures In Systems On Chip

Due to the constantly increasing density of micro-electronic circuits, new systems with hundreds number of processing elements require an efficient interconnection. However, connecting these processing elements led to interconnection problems where the system performance depends largely on the interconnection and the power consumption. Moreover, the traditional interconnections based on shared buses are limited in terms of performance because many elements are not allowed to interconnect in the same time. Based on this observation, several research groups have been working on a new interconnection to adapt the future complex SoC where the NoC is offered as a novel interconnection paradigm. In this section, some interconnection structures will be discussed briefly.

Point To Point Connections

In the point to point connections, the dedicated links are established between each pair of communicated items or all items are linked together as shown in Fig. 2.1. In this case, all nodes are connected to all other nodes by direct links. The advantages of this connection are its speed due to the direct connection and the failure of any link does not effect the other links because each link is independent of other connections. However, the main disadvantage is that the cost in terms of interconnections is high which causes scalability limitation. Also, when any node is busy and can't receive data, the sender keeps waiting. Consequently, it is inconceivable to use this type of architecture when the number of interconnected

units is very large.

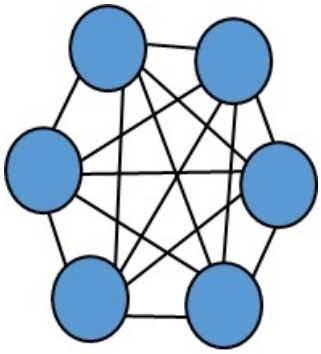


Figure 2.1: Full interconnection.

Shared Bus

In the shared bus topology, each node is connected to a shared bus as shown in Fig. 2.2. The shared busses are much more flexible and better reusable compared to point to point links. However, the main disadvantage of the shared bus is the limitation of interconnection to one connection at a time. Therefore, the bandwidth of each element decreases with increasing number of communicated items. In addition, buses require additional arbitration mechanism to deal with competing demands for access.

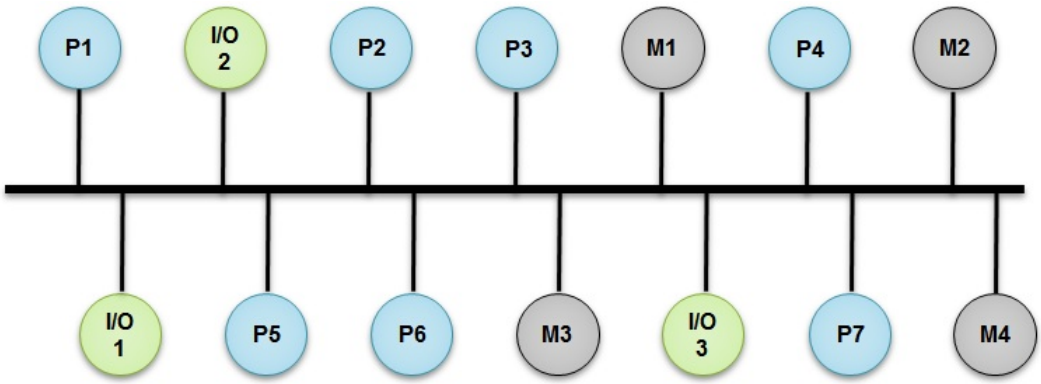


Figure 2.2: Shared bus.

Hierarchical Bus

In order to increase the performance of shared bus, a hierarchical bus is used. A hierarchical bus connects multiple shared buses by using bus bridges as shown in Fig. 2.3. These bridges can serve as a converter protocol if the connected buses use different protocols. However, the connections between buses may require more parallelism, different bandwidth, and protocols that lead to scalability limitation. Hence, the problem of scalability is not resolved.

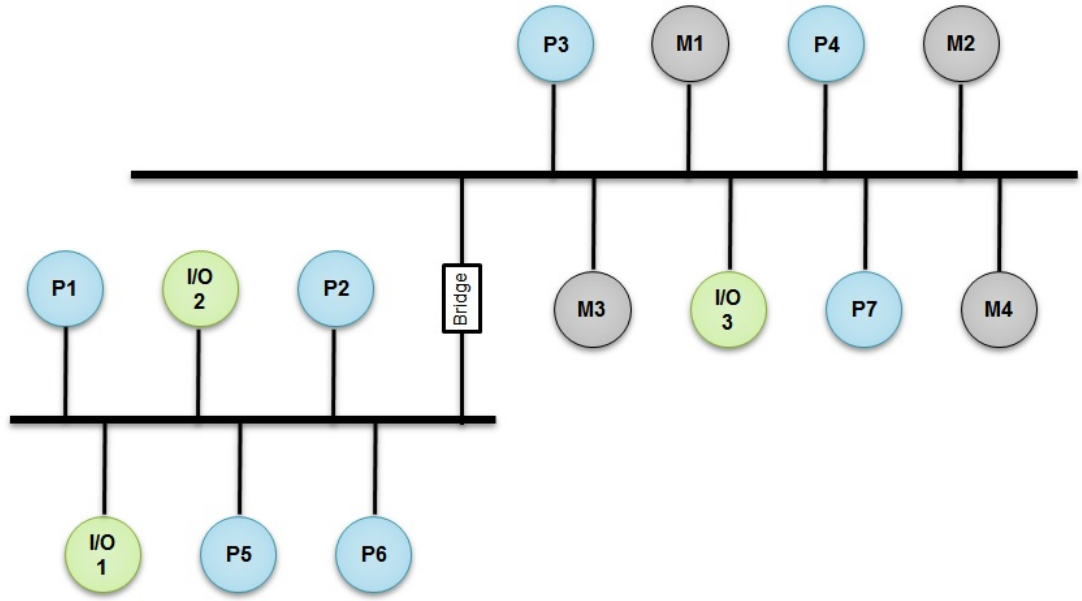


Figure 2.3: Hierarchical bus.

2.1.2 The NoC Paradigm

In the buses interconnection, it is very difficult to add new elements to communicate because firstly, the bandwidth allocated to each element decreases as the number of elements increases, and secondly, the control of bus becomes complex which limits the scalability of this communication structure. Thus, these limita-

tions make them incompatible with the future generations of integrated circuits that include large number of processing resources. These problems have already been studied and the solutions were described in [15, 16] in which the NoC is utilized instead of bus.

Overall Implementation of NoCs

Each NoC has its own characteristics in terms of latency, throughput, area, energy consumption, reliability, etc. These characteristics are directly related to the architectural choices including switching, routing and topology.

The Basic Components of NoCs

The NoC architecture consists of nodes, network interface (NI), routers and communication links as shown in Fig. 2.4. Nodes are consider as the computing elements or memories which correspond to the points of messages injection and ejection in the network. The network interface (NI) is used as an adapter between the nodes and the network. It allows the nodes to be connected and to utilize the network while the routers are used to route the messages from the source to the destination node.

Topologies of NoCs

The NoC topologies are very similar to those of the conventional computer networks. The topology defines how the resources are linked together. Each topology has its special characteristics. The NoCs topologies can be regular such

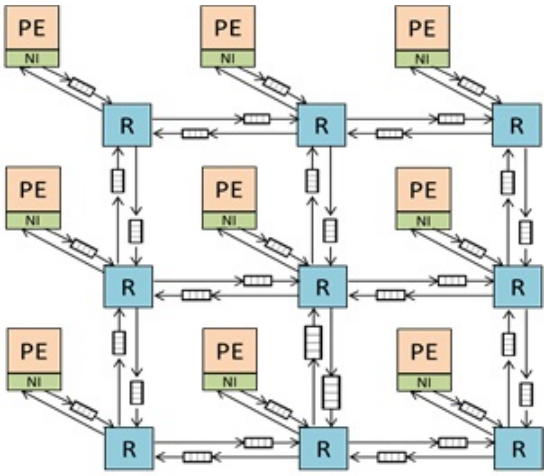


Figure 2.4: Basic components of NoC.

as mesh, torus, and ring, or irregular such as spidergon [17, 18, 19, 20, 21, 22, 23]. In this part, some of these topologies will be explored.

- Mesh and Torus Topologies: A mesh topology is simple and easy to implement. It involves that all nodes are connected with their neighbors and with local node. Therefore, four ports that are east, west, south, and north are needed to connect the router with its neighbors while a local port is needed to connect it with PE. This topology allows some measure of reliability because in case of link or node failure, the other ways are possible. The simplest form of a mesh topology is two dimensions, in which every node is associated with a router, as shown in Fig 2.5 (a).

  A torus topology is derived from mesh topology in which the first element in each row and column is connected with the last element in that row or column to reduce number of hops between the source and destination nodes as shown in the Fig. 2.5 (b).

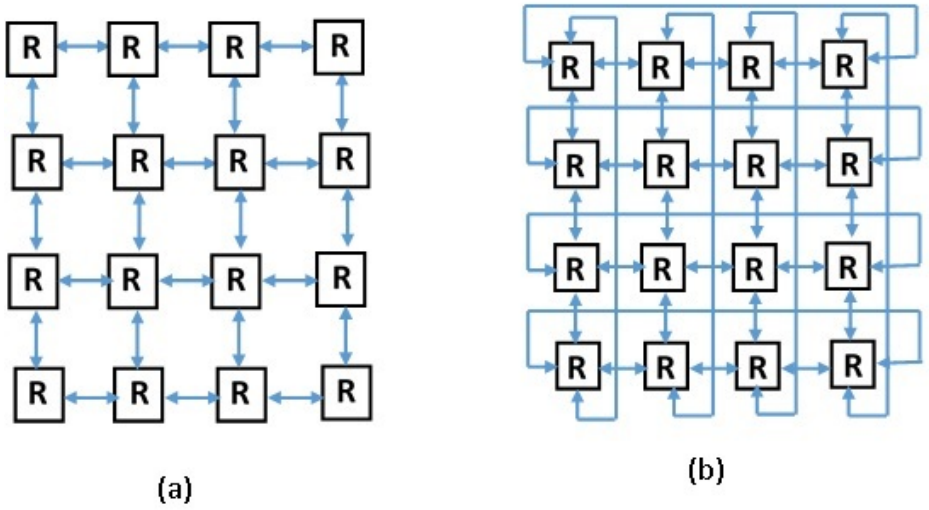


Figure 2.5: Mesh and Torus topologies.

- Ring Topology: The ring topology connects the nodes of a network in a loop form as shown in Fig. 2.6 (a). The ring topology may be unidirectional or bidirectional rings. However, as the number of node increases, the end to end path becomes longer, and the longest path is n, n/2 in unidirectional and bidirectional ring respectively where n is the number of nodes.

- Tree Topology: The tree topology uses the hierarchy and the parent-child relationship. The node at the top can be connected to the multiple nodes at a lower level which can themselves be connected to the other nodes of the lower levels and so on. One risk of this topology is that if one parent node fails, all his children are disabled. Moreover, a derived topology called fat tree where nodes can only be the last level (leaves) shown in Fig. 2.6 (b).

- Star Topology: The star topology connects all nodes via a central connection as shown in Fig 2.7. The advantage of this configuration is that if one node fails, the remaining nodes can continue to operate normally. In the other side,

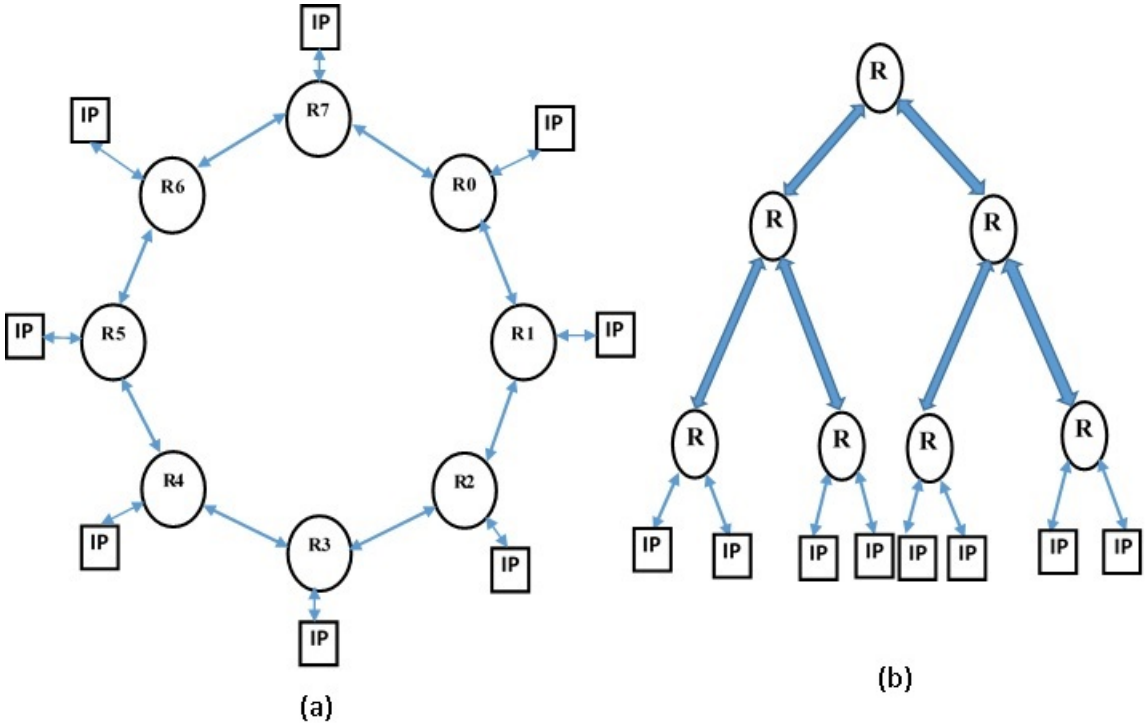


Figure 2.6: Ring and Tree topologies.

the star topology suffers from a single point of failure issue. To illustrate, if the central connection goes down, the entire network goes down too because the data transfers between the nodes can only be made using the central connection.

- Other Topologies: Some architectures were developed by using the features of different topologies [24, 20, 25, 26]. The Spidergon is an example where the nodes are arranged on a bidirectional ring and interconnected by a crossbar as a star structure. Two examples of Spidergon that are Octagon and Polygon are shown in Fig. 2.8. The best path is chosen thorough ring or crossbar.

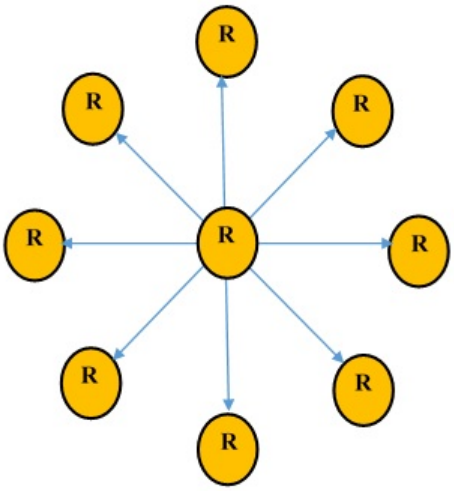


Figure 2.7: Star topology.

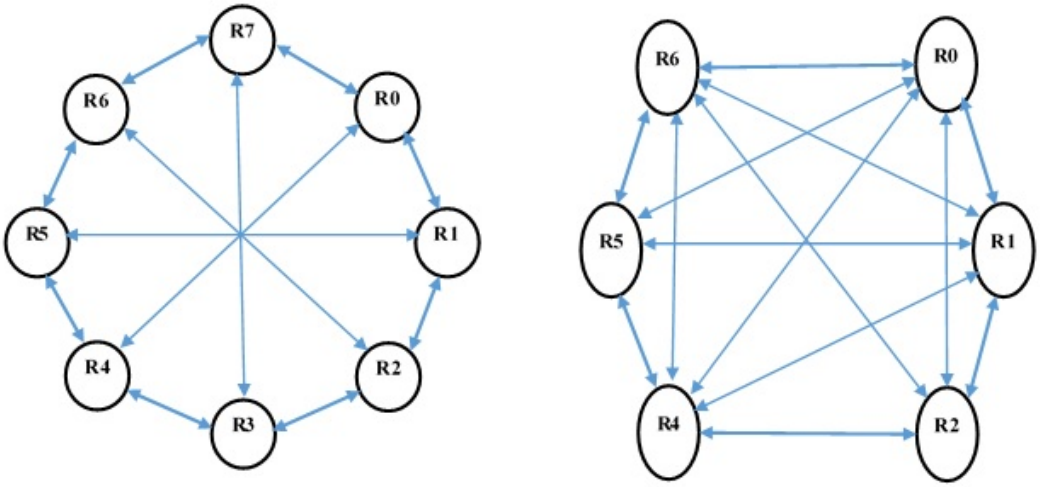


Figure 2.8: Octagon and Polygon topologies.

Switching Techniques

Switching techniques determine when and how the input port of the router can be connected to the output port. The messages are divided into packets which in turn are divided into flow control units (flits). There are two types of switching

techniques commonly used that are circuit switching and packet switching.

- Circuit Switching: In circuit switching, a unique connection is established between the transmitter and the receiver for duration of time required to transfer the data. When the connection is available, an entire message is sent through the same path. In this respect, if there is a conflict with already reserved path during the establishment of the link, the sending of a message is delayed until the conflict is released. The switching technique requires minimum memory and avoids deadlocks.

- Packet Switching: In packet switching, there are no reservation links and the message is divided into packets that are independent of the message in their way to the destination. In this respect, if more than one packet compete for the same link, only one may have it and the others need to wait. There are three main techniques of packet switching which are store-and-forward, virtual cut-through, and wormhole.

    1. Store and Forward (SAF)

       The store and forward is the simplest technique of packet switching. The packet is sent only when the receiver has enough memory to receive the whole packet. The routing information is included in each packet. When a packet arrives at any node, it is stored in that node and then the routing information is extracted from the header to determine the direction in which the packet should be forwarded.

2. Virtual Cut Through(VCT)

   VCT is similar to the previous technique because the destination must be able to receive the whole packet. The main difference is that the routing information is taken from the first byte of the packet and the packet is sent as soon as the direction (output) is determined. Hence, only fraction of packet is stored in each router's buffer, and then the fractions are forwarded to the next nodes. This allows for faster data transfers because the communication latency is reduced compared to SAF switching.

3. Wormhole (WH)

   In WH, the packets are divided into smaller parts called flits where the first flit is called a header flit and it contains the control data. This technique differs from the virtual cut-through in that it requires storage space for only a flit instead of a whole packet. Consequently, WH achieves the best performance in terms of memory requirements and latency but deadlock; several packets block each other due to a cyclic dependency; is more likely to happen.

Routing Algorithm

Routing algorithms determine the path from the source to the destination and allow to manage the transmission of packets within a network. It regulates the traffic and determines the path in which a packet is forwarded to reach its destination. There are many types of routing algorithms based on network specifications

and the performance to be achieved [27]. This section explores two major types of routing algorithms within NoCs which are deterministic or adaptive algorithm and source or distributed algorithm.

- Deterministic or Adaptive Routing: deterministic routing algorithms operate on the principle of a fixed decision. A packet generated at node A and required to reach node B always follows the same path regardless of the network status. This class of algorithms includes the routing functions that combine a unique route between each pair (source, destination). According to the network topology, it is possible to implement the deterministic routing by routing tables or by a sequence of arithmetic operations based on the source and destination address of each message. The advantage of the deterministic routing is its easy implementation which results in a very limited number of hardware resources. The most frequent static algorithms used is a Dimension-Ordered-Routing (DOR) and the most common is XY routing used in mesh topology. The principle is simple, the packet is routed on the dimension on which there is a minimum distance (relative to the destination). The advantage of static algorithms lies in their simplicity of implementation. However, the lack of flexibility in the routes allocation makes them very sensitive to congestion.

  Unlike deterministic routing, adaptive algorithms allow more freedom. These algorithms take into account the network congestion state or the states of their closest neighbors. When a packet is sent, a request to adjacent node

can be sent and the decision depends on the available path. The adaptive algorithms are more efficient in the networks that suffer from congestion or damage. On the other hand, this type of algorithm is more complex.

- Source or Distributed Routing: In a source routing, the node from which the packet is generated chooses the path in which the packets take to reach their destination without further calculation during transmission. The source routing is perfectly suited for a given topology where each node communicates only with few specified nodes. This is because the nodes need to send data only through the predefined paths. For instance, in the processor system with a single memory, each processor can send a packet to the memory via a predefined path and already encoded. The drawback of this technique is that it requires a larger header size that results in a lower bandwidth utilization. In distributed routing, the node has to store the packet before transmitting it until the congestion is solved. To do so, a router decides which path is the most appropriate to forward the packet. Unlike source routing, the starting point is not required to know the entire path that the packet goes through. This type of routing is appropriate in the case of congestion where a node can modify the route of a packet based on the available path toward the destination. The disadvantage of a distributed routing is that it requires more computation circuits compared to the source routing.

Flow Control Protocols

Flow control protocols are designed to ensure a safe transmission of packets from source to destination by controlling the flow of traffic between them. The concept of flow control guarantees that no packet will be lost due to buffers overflow [1]. Thanks to flow control, the transmitter will not send more data than the receiver capacity. This control can be applied between adjacent routers and/or between transmitters and receivers. The flow control can be classified into two main types:

- Buffer-less Flow Control: The simplest forms of flow control does not use buffering and simply allocates channel state and bandwidth to competing packets. In these cases, the flow-control method must perform an arbitration to decide which packet grants the channel it has requested. Since there is no buffer, the ungranted packet will be dropped [1].
- Buffered Flow Control: In this approach, the packets that cannot be routed via the desired channel are stored in buffers. The buffered flow control can be classified into three main types:

  On-Off Flow Control

  In this type of flow-control technique, a single bit is switched between on and off depending on the threshold level in the downstream. An off signal is sent back when the number of flits goes below the threshold while an on signal is sent back when the number of flits rises above the threshold.

  ACK / NACK Flow Control

  This protocol is a very simple protocol where the logic of flow control is

responsible for controlling the traffics between two adjacent routers. The sender sends a request signal to the next router in the path and waits for the acknowledgment (ACK) signal from the receiver. As an acknowledgment is sent back from receiver, the sender sends a packet to the receiver in which it is written in the local buffer while NACK is sent back if there is no available space at the receiver side as shown in Fig. 2.9.

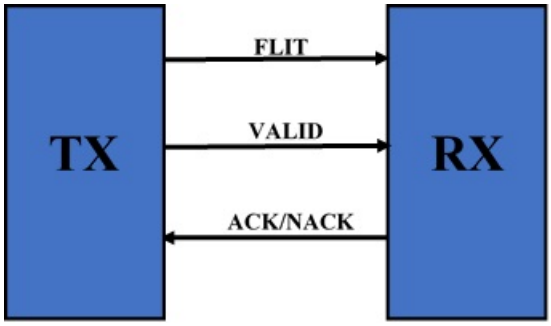


Figure 2.9: ACK/NACK flow control.

Credit Based Flow Control

The main concept of a credit-based communication is to prevent the overflow of the buffers. In doing so, the free space buffer in each downstream virtual channel is tracked by upstream router by keeping a count of free flit buffers in each virtual channel. Then, the count is decremented each time the upstream router sends a flit to downstream router. The upstream router stop forwarding a flit when the count reaches zero (no buffer space in the downstream router). Once the downstream router forward the flit, it increments its counter and sends a credit to the upstream router to increase its counter. This approach guarantees that the movement of a flit is done when there is space in the next router as shown in Fig. 2.10.

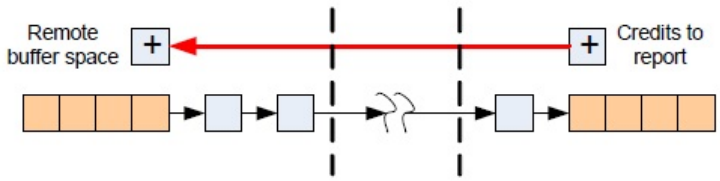


Figure 2.10: A general crediting scheme.

Deadlocks and Livelock

Deadlocks occur when several packets block each other due to a cyclic dependency. Each node is then waiting for sending a packet to a node that is already pending. The wormhole switching with virtual channel is highly eliminating this type of deadlock. In the ring topology, the dependency occurs as each packet holds a resource (i.e., buffer) while requesting another resource (i.e., downstream buffer) and results in a routing deadlock [28].

Live locks occurs when a packet never reaches its destination while it is continuing to travel on the network. The live lock is avoided in the ring network because the packet is either forwarded toward the destination or toward local node.

Congestion Control

Congestion occurs when multiple packets within a router request the same output port simultaneously. Congestion decreases throughput and increases latency in the network. In order to limit or eliminate congestion, different congestion control techniques have been proposed where the simplest one solves the problem by removing packets that cause the congestion, but it introduces an increase in the latency. Another solution lies in the dynamic routing that ensures to avoid the

congested areas of the network or in reducing the packet injection rate in the network to relieve congested areas.

Buffers

Buffers represent an important part in the NoC design because it has an impact on the area, power, and performance. Buffers can be used in the input port, output port, or both to increase the NoC performance. On the other hand, increasing buffer size results in power and area overhead. A trad-off should be taken into account when designing buffers in a NoC.

Virtual Channels

Virtual channels are multiple buffers associated with one physical channel. They are utilized in the input port only, output port only, or in both. The arrived packet is stored in the available virtual channel of an input port waiting for an available output port while the output buffers store the packets that are waiting for available input port in the next router. There are many advantages of virtual channels: firstly, they are used to avoid deadlock. Secondly, virtual channels are utilized to facilitate the use of adaptive routing algorithms. Thirdly, virtual channels also provide a reduction in the number of necessary connections and increase performance. [1] shows that performance can be improved by dividing the size of buffer into several virtual channels. Finally, virtual channels can be used for separating the traffic and offer different levels of priority.

Router Micro-architecture

The construction of router pipeline that represents the flow control scheme and a routing logarithm is shown in the Fig. 2.11.

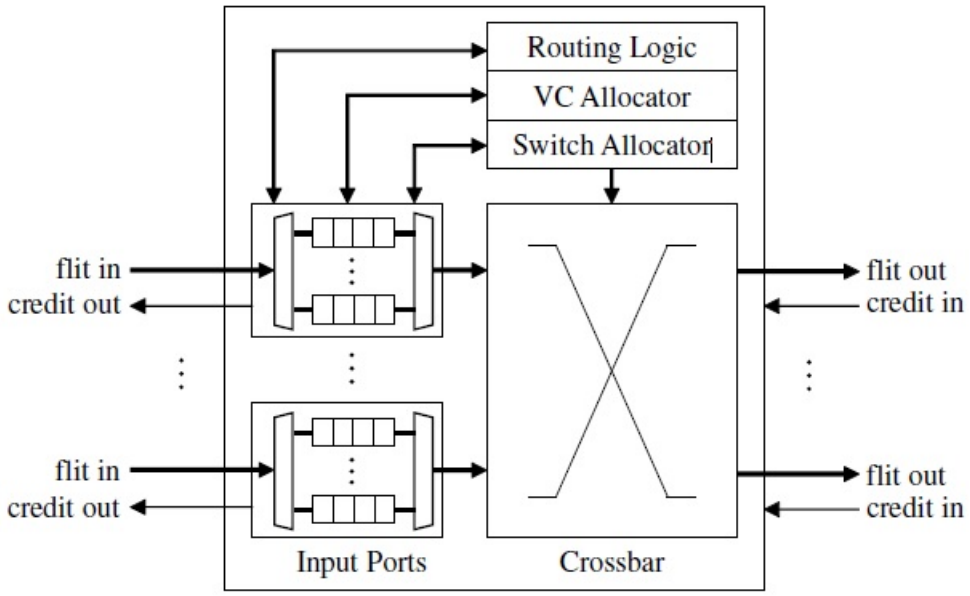


Figure 2.11: Architecture of classic credit based wormhole virtual channel router.

The basic components of the router are : input ports that include the virtual channels, the routing computation module (routing logic) to select an appropriate output port, output port, virtual channel allocator, switch allocator, and crossbar that connect the input port with the output port. As a flit arrives to an input port, it is stored in one of the virtual channels. Then the routing entity uses the routing information included in the header flit to compute the routing path for the incoming packet. After that, the output port is selected by a switch allocator. At each time, a credit is sent to the previous router and received from the next router.

The NoC performance is mainly dependent on the routing mechanisms, topology types, router architecture and flow control mechanisms. Thus, these components should be modeled carefully in the simulators to give an accurate estimation of the network performance [23].

Traffic Models

Traffic models represent the behavior of a system. They are used to measure the different NoC metrics such as performance and average packet latency. In addition, traffic models can be either realistic or synthetic. In a realistic traffic model, a trace of real application executed on NoC is read from input file that includes this information while in synthetic traffic model [1], the traffic is generated using mathematical equations. Moreover, the synthetic traffic gives a good estimation of a NoC performance under the worst-case traffic. In the following subsection, some synthetic models will be discussed briefly.

- Uniform Traffic Model: In this model, each node sends its packets to uniform random destinations. This model is suitable to evaluate the NoC.
- Transpose Model: In this model, the destination address is the transpose of the source address.
- Bit Reverse: In this model, the destination address is the reverse bits of the source address. For example, if a source address is

$$n = n_1, n_2, n_3, ...n_m, \tag{2.1}$$

the destination address is

$$B(n) = n_m, n_{m-1}, n_{m-2}, ...n_1. \tag{2.2}$$

- N Complement: The destination address is the complement of the source address. For instance, if a source address S=0,1,2,3,4 then the destination address D=4,3,2,1,0. The complement here means the summation of source and destination address is equal to network size -1.

NoC Performance Metric

The NoC performance metric is strongly related to the network architecture itself. Thus, it is essential to have a good understanding of these metrics and how to evaluate them to optimize the network architecture.

In this section, two metrics used to evaluate NoC performance will be discussed:

- Latency : latency is one of the most important criteria for the characterization of NoCs. It is measured by time (usually expressed as the number of cycles) necessary to forward a packet from a source node to a destination node. To clearly define the concept of latency, it is necessary to define the format of packet transmission. Packet consists of several flits (Flow Control Units) where the latency can be defined as the number of cycles from the injection of the first flit (header flit) until the receiving of the last flit by the destination. In more general case, the latency is expressed as a function

of the injection rate, congestion, and virtual channel. The average packet latency can be expressed by the equation in [29]:

$$L\_avg = \frac{\sum L_i}{P}, \tag{2.3}$$

where L_i is the latency of ith packet, and P is the number of received packets.

- Throughput: throughput reflects the amount of data transferred during a period of time. Throughput can be measured in packets per clock cycle or packets per second. It can be normalized to be independent of packet size by dividing the previous value by the size of the packet and network size . In this case, the unit of throughput will be in bits per node per second or clock cycle. The throughput can be defined as follows [30]:

$$Throughput = \frac{Number\ of\ Recieved\ Packets\ *\ Packet\ Length}{Number\ of IPs\ *\ Time}, \tag{2.4}$$

where *Number of IPs* represents the number of intellectual properties that send the data over the network and *Time* is the length of interval, in clock cycles, between the generation of the first packet and the reception of the last packet.

2.2 State of the Art In NoC Simulation

NoC simulators can be classified into three categories; 1) SW-based, 2) FPGA-based, and 3) analytical models-based. There is considerable work in the field

of analytical models of NoCs that provide generalized models for different NoC router architectures [31, 32, 33, 34]. These analytical models, however are not suitable to be used within execution driven architectural simulators due to their static nature. Below is a review of the state of the art in SW and FPGA-based NoC simulators.

2.2.1 NoC Software Simulators

A team of Computer Architects at the University of Catania (Italy) developed a cycle accurate NoC simulator called Noxim using SystemC. The latest version of Noxim [35] supports any topology, many routing and network configurations including hubs (for none adjacent tiles such as in wireless hubs). It is in SystemC so it can be integrated with an architectural simulator. It supports various abstraction levels from cycle accurate to transaction-level-modeling. It collects performance and power data. Its main problem is simulation speed and it is as fast as Booksim at best. For larger NoCs or higher injection rates Booksim is actually significantly faster. GARNET is a detailed cycle-accurate NoC simulator [2]. It can be used as a stand alone or integrated into the full system simulator GEM5 [36]. The main limitation of GARNET is its individual network component configurations. In [4], SICOSYS was proposed as a cycle accurate Network-on-Chip simulator and specifically target multiprocessor systems. Although SICOSYS has been incorporated into [37] for simulating symmetric multiprocessor systems, the platform cannot be used to study CMP systems with a NoC. In [5], the Booksim

simulator was proposed as cycle accurate Network-on-Chip simulator and can be used to model the interconnection networks for a variety of other systems. Thus, the simulator is considered as a general purpose interconnection software-based NoC used to implement different functionalities of the NoC. It can be used to evaluate different aspects of NoC design such as topology, flow control, routing technique, router architecture, and quality-of-service.

2.2.2 FPGA Based NoC Simulators

An early FPGA-based NoC simulator was reported in [38]. It implemented a 2D torus topology with wormhole routing mechanism but it suffered from large area. The work was later extended in [39] but it still suffered from a large area. In [40], a 3x3mesh topology was implemented with a store and forward mechanism, and XY routing with virtual cut through flow control . However, it consumes a large area for FPGA platforms.

In [41], a parameterizable router with variable buffer depth and data width was implemented. A mesh topology with size 5 x 5 is simulated to explore the effect of buffer depth. The authors concluded that the increasing of buffer size reduces the packet latency. However, this trend is limited by saturation point. Also, the design suffers from a very low clock speed. The work in [41] was later enhanced in [42] to support an analyzer of traffic and an automatic router generation. RASoC [43] was presented with a wormhole flow control. The frequency used was almost 57 MHz and the area occupied by the router is reasonably large

with limited buffers to 4 buffers per port. In [44], PNoC was proposed with a circuit switching flow control. The complexity of routing increases as the number of ports increases. Hence, the scalability of design is reduced and also, the design suffers from additional latency due to the circuit switching setup and tear down delay and also blocked communication due to the possible idle time. In [45], GNoC is proposed as a generic router that provides a variety of switching, arbitration and routing mechanism. The tool was developed to explore the sharing of many decentralized components to decrease area.

In [10], the DART NoC was proposed. In this simulator, NoC can be virtualized by mapping NoC parts to a generic engine of NoC simulator. DART can simulate different NoCs with no hardware modification of the simulator on the FPGA since the parameters can be set at run-time by a software. To do so, DART architecture decoupled DART cycles from the cycles of the simulated system using a global counter. Also, it decoupled the simulators architecture from that of the simulated NoCs architecture by using a global MUX-based interconnections. Although, DART divides the nodes into partitions and uses a crossbar for partitions to reduce the cost of global interconnection, it is difficult to simulate large NoC designs. This is because the cost of crossbar area increases quadratically with respect to the number of input and output ports. Also, for large NoC designs, the routing table will be too large.

In [12], FIST was proposed. FIST estimates NoC packet latencies in the context of full-system CMP simulators. Each router in the network is modeled

as a set of load-delay curves. According to the degree of accuracy, more than one router can be modeled as a one load-delay curve. The load-delay curves are generated online using training or off-line according to a traffic pattern and the network configurations. The packet latency is estimated by using load-delay curves at the router in which the packet traverses. The NoC model is made of a single packet FIFO and an array of routers models. It is not clear if this can be integrated with a full-system architectural simulator. Due to the high abstraction level used in FIST , it is limited to specific types of network and traffic patterns. FIST also utilized BRAMs extensively to store the load-delay curve which limits its scalability. Furthermore, FIST cannot be used with adaptive networks and with networks that encounter higher injection rate than that used in the training rate. Moreover, the accuracy of FIST is very sensitive to both buffering and load where the buffering and traffic load have their effects on the network performance such as average packet latency and throughput. Hence, as load increases, the congestion is built up in some routers that directly affect other adjacent routers which leads to losing the accuracy in FIST. In short, the good environment for FIST is a network with low load and limited buffer. In [11], TDM approach was utilized to emulate a 128x128 mesh. It eliminated the need for large memory in the source queue by storing the time stamp for the packet in the source queue instead of full packet. Due to the use of time-multiplexing, the emulation speed is very low. In [9], an HW-based NoC simulation platform was proposed. It is composed of two boards: a SOC board and an FPGA board. The SoC board includes

two ARM9 processors while the FPGA board includes a virtex-II8000. The TDM mechanism is used to simulate all routers in the network sequentially by using one router. Moreover, generating the network traffic, controlling the network router, and analyzing the output packets are done by the ARM9 processor. However, the off-chip communication represents a performance bottleneck.

The main constraint in FPGAs is the resources that limit the size of the design. Using shared resources decreases the area of the design but also the performance. In [46, 47], a shared VC buffer among adjacent ports is used to reduce the area of the design. This sharing, however increased the failure probability. This problem was solved in [48] by separating the control units of the shared resources but the area overhead still prevailed. Many of the above-mentioned FPGA-based simulators share similar features and most can not be attached to an architectural simulator because this was not intended from the beginning. In contrast, FBNoC was developed with this in mind; it should be integratable with other architectural simulators, deliver the packets to their destinations, and not just compute packet latencies/throughputs.

CHAPTER 3

THE PROPOSED FBNOC SIMULATOR

3.1 Simulator Overview

The proposed FBNoC can simulate four NoC topologies that are mesh, torus, ring, and fat-tree. The top level architectural view of the proposed FBNoC simulator is shown in Fig.3.1. FBNoC is basically a ring of nodes connected to adjacent nodes by two bidirectional links (i.e., it is two bi-directional ring NoCs). As can be seen from this Figure, each NoC node can be connected to one or more cores (i.e., clients) through bidirectional local ports (LPs). In addition to LPs, each node level is composed of two main parts; a latency model and a router. When used in stand alone mode (i.e., just simulating a target NoC), a traffic manager is added to the design.

Fig. 3.2 illustrates how FBNoC can be integrated with a full many-core

architectural simulator for different number of cores/NoC node. It shows how FBNoC can be integrated with the typical 2D tiled architecture of a many-core simulator. The floor plan of the FBNoC with such a simulator is shown for three different scenarios; one NoC node/core (i.e. one-local port per NoC node) Fig. 3.2 (a), two local ports per node Fig. 3.2 (b), and four local ports per node Fig. 3.2 (c). To use FBNoC in stand alone mode (i.e. with no cores), the local ports are replaced with traffic managers. Next, each component of the FBNoC simulator node is described in details.

3.1.1 Traffic Manager and Latency Model

When FBNoC is used in stand alone mode, the traffic manager is responsible for injecting/ejecting packet into/from the network. It reads the network configurations from a BRAM (block memory) that contains the user-specificied network configuration and design parameters such as topology, number of virtual channels per router, packet size, injection rates (for synthetic traffic), ...,etc.. The traffic manager inject packets into the NoC based on these parameters. The traffic manager also serves as a traffic sink. It ejects packets from the network and deliver the source-destination addresses to the latency calculation model, which in turns compute the packet latency and output it to the traffic manager. The traffic manager adds a time stamp to the packet based on the computed latency and collect different performance measures such as average packet latency and throughput. The

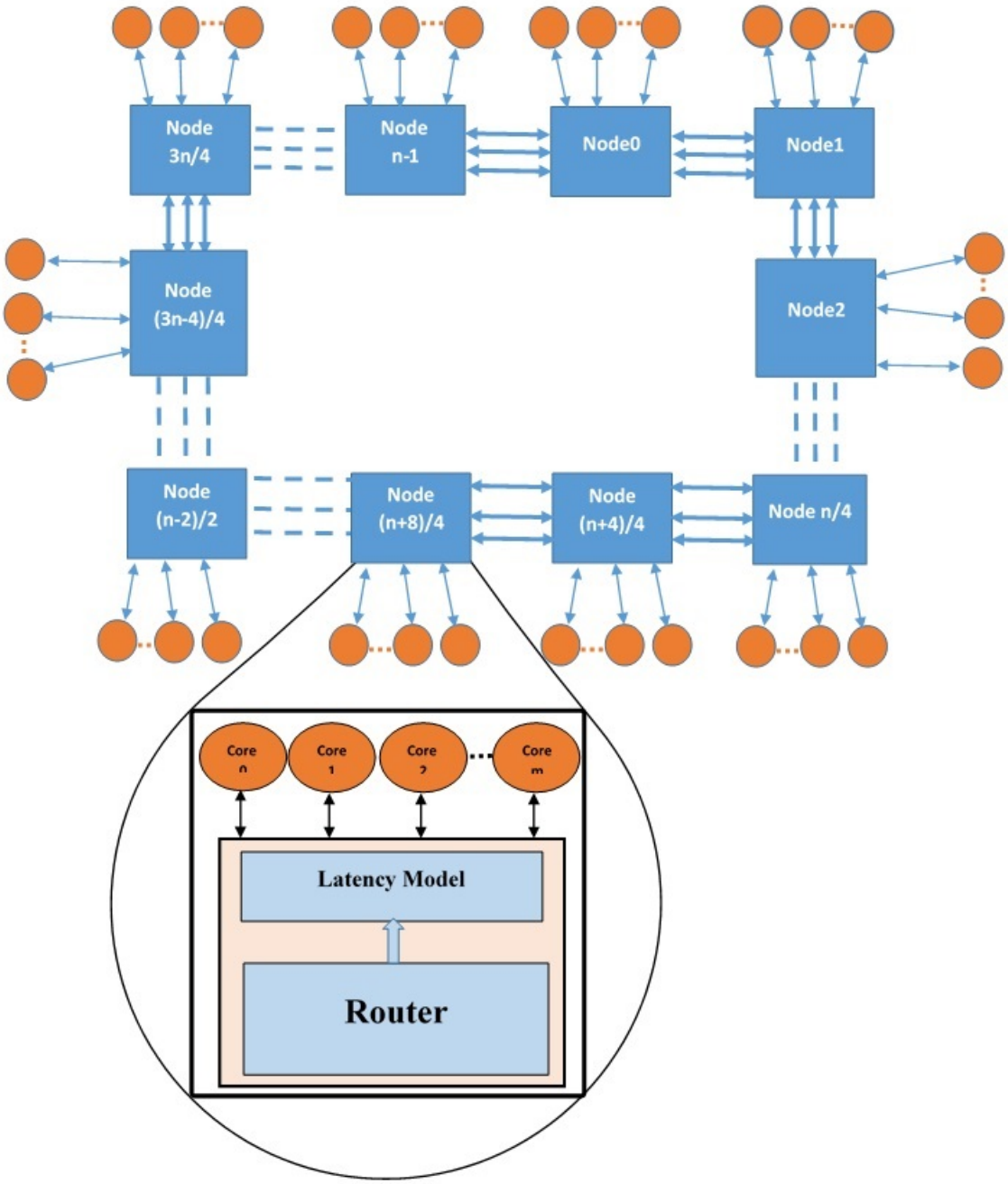
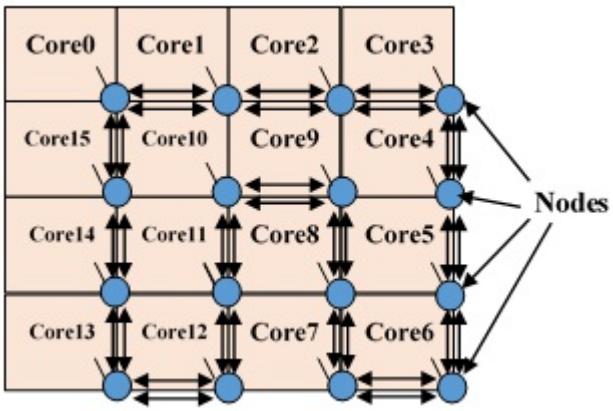


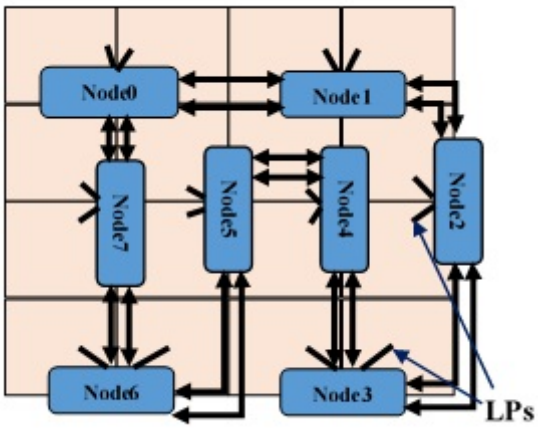
Figure 3.1: Top and node level view of the FBNoC simulator's architecture.

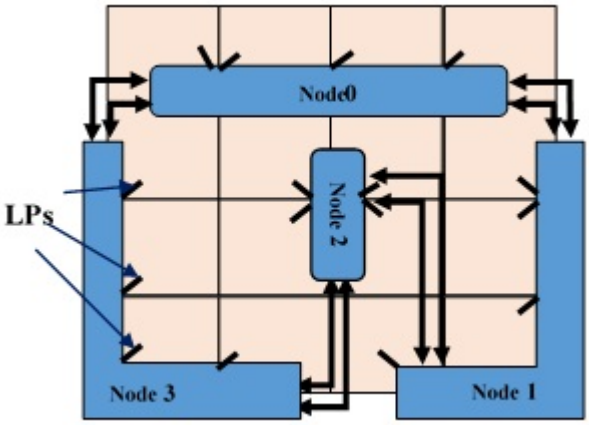
latency model calculates the packet latency depending on the user-specification configurations delivered by the traffic manager. The packet latency is a function of topology, number of virtual channels, injection rate, number of hops, and con-

(a)



(b)





(c)

Figure 3.2: The floor plans of FBNoCs integrated with a many-core architectural simulator on a single FPGA chip. (a) One local port per node. (b) Two local ports per node. (c) Four local ports per node.

gestion per hop. The block diagram of latency model is shown in Fig. 3.3. The main components of the latency model are the hop counts and latency calculation. A congestion regression model is utilized to estimate the average congestion per hop and is only included in node 0. Furthermore, the proposed regression model consists of an adder-multiplier circuit and memory to store the coefficient values that can be used later in regression model. The estimated average congestion per router (ACPR) value is forwarded to all Nodes at regular interval using additional parallel ring.

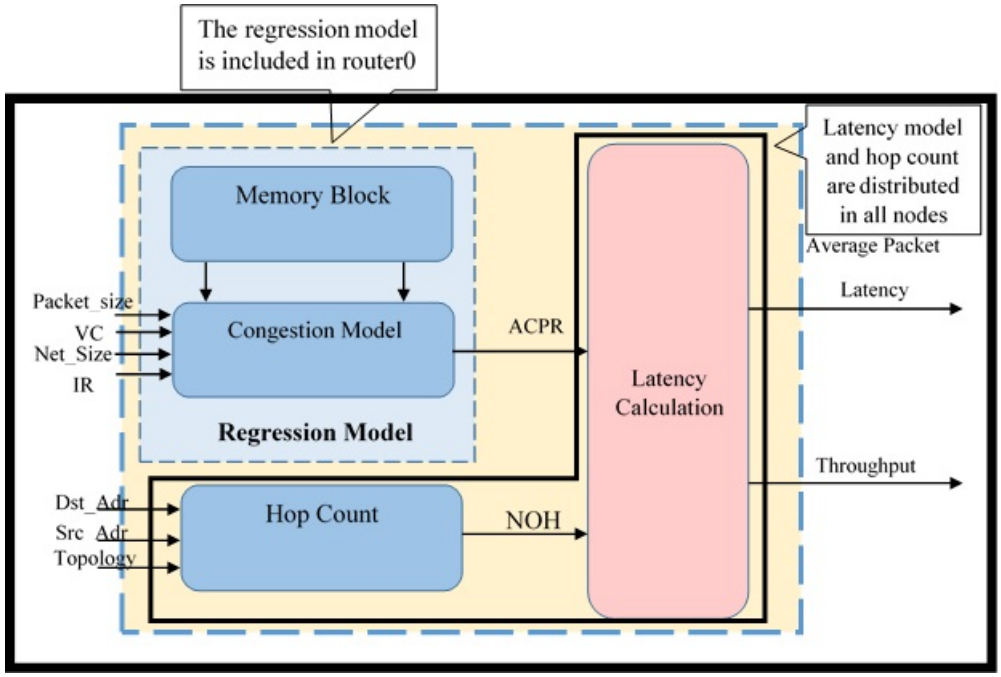


Figure 3.3: The latency model block diagram.

Hop Count Module:

The hop count module is used to calculate the packet hop count that represents the number of routers in which the packet traverses through from the source node

to the destination node. It depends on the NoC topology and is calculated using the source and destination address of the packet. The current version of FBNOC supports ring, mesh, torus, and fat tree NoC topologies. For mesh and torus topologies, if the source and destination addresses coordinates are Si, Sj, Di, Dj then Δy and Δx (hops in x and y directions, respectively) are obtained in [1] as follows:

$$\Delta y = |D_j - S_j| \,, \tag{3.1}$$

$$\Delta x = |D_i - S_i| \,. \tag{3.2}$$

The total hop count can then be calculated in [1] as follows:

$$Hop\_count = \Delta x + \Delta y. \tag{3.3}$$

In torus, they are obtained in [1] as follows:

$$\Delta y = \begin{cases} |D_j - S_j| \,, & |D_j - S_j| \ <= k, \\ \\ k - |D_j - S_j| \,, & |D_j - S_j| \ > k, \end{cases} \tag{3.4}$$

$$\Delta x = \begin{cases} |D_i - S_i| \,, & |D_i - S_i| \ <= k, \\ \\ k - |D_i - S_i| \,, & |D_i - S_i| \ > k. \end{cases} \tag{3.5}$$

The total hop count can then be calculated as:

$$Hop\_count = \Delta x + \Delta y \tag{3.6}$$

where k is the number of routers per dimension.

For a Fat Tree NoC, the hop count can be obtained as:

$$Hop\_count = 2L - 1, \tag{3.7}$$

where L is the lowest common ancestor which is the index of the highest significance bit difference between the source and destination addresses. e.g. for a network of size 64, sending a packet from core 0 (address 000000) to core 63 (address 111111), the highest significance bit difference is the sixth bit, so L=6. Finally, for bidirectional ring topology, the packet can be forwarded either in clockwise direction or in counter clockwise direction depending on the shortest path to the destination. Hence, the maximum hop count from the source to destination is network-size /2. If the difference between source address and destination address is

$$\Delta r = |dstenation - source|, \tag{3.8}$$

then the hop count can be obtained as follows:

$$Hop\_count = \begin{cases} \Delta r, & \Delta r \quad < net_{size}/2 \\ \\ net_{size} - \Delta r, & \Delta r \quad >= net_{size}/2 \end{cases} \tag{3.9}$$

Latency Regression Model:

A multi-variable linear regression model is used to predict the latency of each received packet in cycles. This is the sum of times it takes a packet to go through each router. It includes the congestion time; the time (in cycles) a packet waits inside the router for available output port. Such a model for a dependent variable Y and independent variables $X_1, ..., X_n$ in [49] looks as follows

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \epsilon, \tag{3.10}$$

where β_0 represents the model interception, β_i is the coefficient of the variable Xi, and the ϵ is the regression error. The advantage of the linear regression is being easy to implement in FPGA since it consumes few resources (adders and multipliers as seen in (Eq. 3.10)). Since Booksim's accuracy is generally accepted by most NoC designers and researchers, it was used to carry out extensive experiments to measure average packet latency for different topologies for various injection rates, packet sizes, and number of virtual channels. After that, a descriptive analysis was carried out on the collected results to identify the general behavior therein.

In our model, we collect the data and extract the congestion per hop by comparing the minimum packet latency (zero-load) with the measured average packet latency per hop. Since a zero-load represents the minimum packet latency (i.e, without congestion), the additional latency is then due to the congestion. Hence, the congestion can be obtained by subtracting zero-load from measured packet latency. In addition, the average congestion per hop can be calculated by dividing the congestion over the hop count as follows:

$$Avg_{cng} = \frac{APL - Zero\_load}{avg_{hop\_count}}, \tag{3.11}$$

where APL is the average packet latency taken from Booksim, Zero_load is the minimum latency (no congestion) and avg_{hop_count} is the average number of hops from source to destination. The zero-load latency can be estimated analytically using the formula in [1]:

$$Zero\_load = H_{avg} * D_{hop} + P_{len} - 1, \tag{3.12}$$

where H_{avg} is the average hop count of the network, and D_{hop} is the number of pipeline stages in the router architecture and P_{len} is the packet length. For example, in two dimensional mesh topology, the average lowest hop count in mesh can be obtained by the following formula in [1]:

$$H_{avg} = \begin{cases} \frac{nk}{3}, & k \quad even, \\ \\ n(\frac{k}{3} - \frac{1}{3k}), & k \quad odd, \end{cases} \tag{3.13}$$

and for torus the average lowest hop count can be obtained using the formula in [1]:

$$H_{avg} = \begin{cases} \frac{nk}{4}, & k \quad even, \\ \\ n(\frac{k}{4} - \frac{1}{4k}), & k \quad odd. \end{cases} \tag{3.14}$$

In this work the zero-load latency values were obtained using the load-latency curves generated by Booksim simulation as shown in Fig. 3.4. For most cases, the obtained values were very close to the values estimated using (3.12).

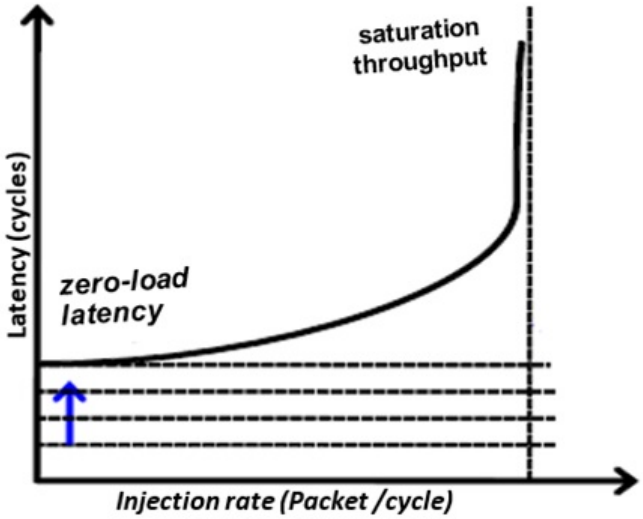


Figure 3.4: Load delay curve.

The obtained average congestion values by Eq. 3.12 can be utilized to obtain the regression parameters. Furthermore, one fitting curve for each network size is

derived and then it can be used to estimate the congestion for different number of virtual channels, packet size and injection rates. The fitting expression for congestion model is a function of average injection rate , packet length, and number of virtual channels. Thus, it can be represented as follows:

$$A_{cng} = (a * AIR + b * (P_{len} - 1) + c * VC + d), \tag{3.15}$$

where the A_{cng} is the average congestion in cycle, AIR is the average injection rate (packet/cycle/node), P_{len} is a packet size in flits, VC is the number of virtual channel, and a,b,c,d are the coefficient values. Since the coefficient values are stored in memory, the regression model circuit is implemented in hardware as a single shared circuit and it can calculate the congestion for different topologies and network sizes (with out re-synthesis or re-configuration). Furthermore, all calculations were normalized to integer values such that only integer adders and multipliers are used to predict the congestion. This improved the performance and reduced the resource utilization significantly. More importantly, the regression formula is very simple and can be used for different topologies and different network sizes.

As discussed before, the zero_load in Eq. 3.12 is utilized to obtain the latency without congestion. To add a congestion to this equation, it can be rewritten as follows:

$$Pkt_{Latency} = R\_D * hop_{count} + A_{cng} * hop_{count} + \frac{P_{len} - 1}{BW}, \tag{3.16}$$

$$Pkt_{Latency} = (R\_D + A_{cng}) * hop_{count} + \frac{P_{len} - 1}{BW}, \tag{3.17}$$

where the R_D is an input parameter that represents the minimum number of cycles required for the packet from arriving to router till departing from the router (i.e. number of pipeline stage), and A_{cng} is the average congestion calculated by the congestion regression model. Eq. 3.17 above is utilized to calculate the per packet latency that is added to the packet's time stamp by the node's latency model. The aggregated average packet latency for all received packets by a node is the summation of latency for all received packets divided by total number of received packets as follows :

$$APL = \frac{1}{NoP}(\sum(R\_D + A_{cng}) * hop_{count} + \frac{P_{len} - 1}{BW}), \tag{3.18}$$

where NoP is the number of received packets.

3.1.2 The FBNoC Simulator Network

The proposed network is two bidirectional rings network with multi-local port as illustrated in Fig.3.5. The main function of this network is to increase overall system throughput, cooperate with architectural simulators, and deliver the packet to its destination where the packet contains one or more flits. Moreover, the size of all input and output ports corresponds to the size of flit. Since the packet latency in the simulator NoC is decoupled from that in the simulated target NoC, our objective was to reduce the latencies in the simulator NoC as much as

possible. Hence the use of two bi-directional rings for packet delivery and a separate bi-directional ring for sending congestion messages from all nodes to Node 0 and to broadcast the current A_{cng} from Node 0 to all other nodes in FBNoC.

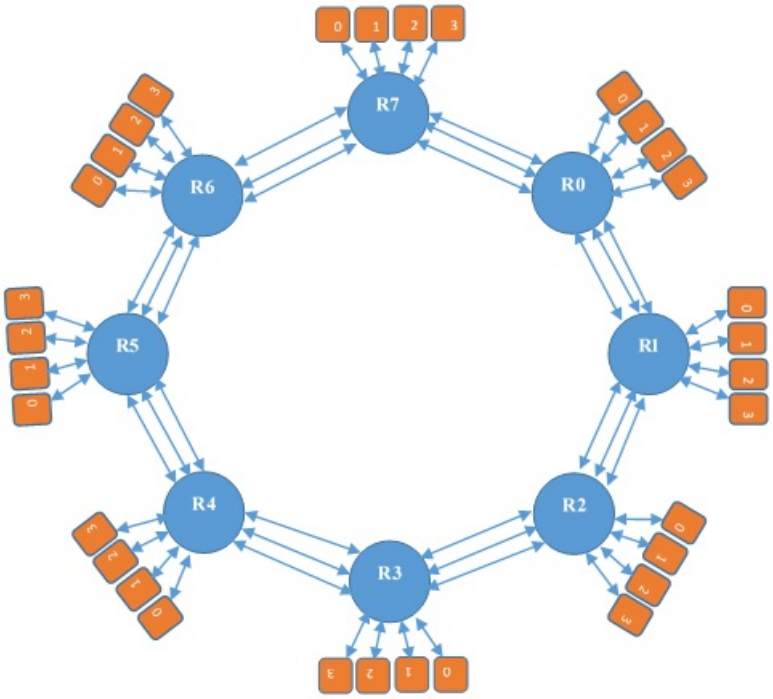


Figure 3.5: FBNoC network.

Router Overview

The NoC separates the communication and computation where the wormhole flow control is utilized in NoC to reduce the memory usage. The packet is divided into flits and enabled to be transferred serially through the network (flit after another). Furthermore, the virtual channels are used to prevent a head of line blocking and avoid a deadlock when the network traffic is heavy that results in improving the overall performance. However, if these features are added to the design, it makes it complex and requires more stages to deliver the packet from the

input port to the corresponding output port which in turn increases the packet latency. To reduce the average packet latency, many stages should be implemented in parallel such as switch allocator, virtual channel allocator, and routing computation. The router has three bidirectional ports that are port1,port2, and local port where port1 and port2 are utilized to form two bidirectional rings and the local port can be multiple local ports to connect IPs.

The router has five main components which are router buffer, input controller, arbiter, output status, and crossbar as shown in Fig.3.6.

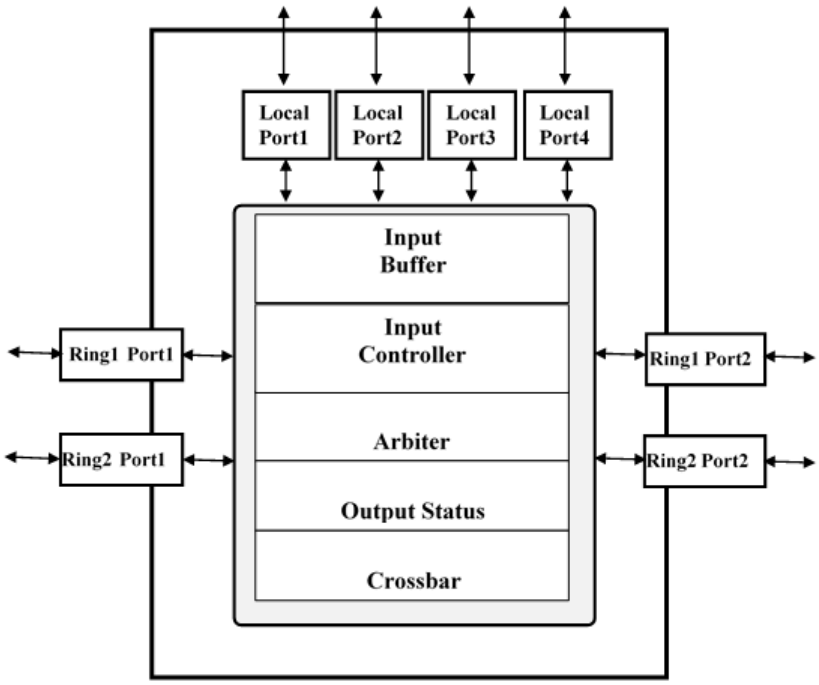


Figure 3.6: Block Diagram of Router

The block diagram of the router consists of the following:

- Router Buffer: The router buffer is created at the input port to store the incoming packets where it has two status signals full/empty to control the

inter/intra-router communication. Furthermore, the input buffer size and the data width are configurable and can be configured by the user.

- Input Controller: The input controller is responsible for implementing a simple ACK/NACK-based flow control. If the buffer's *full* signal is high (buffer is full), the packet can not be received from the previous router, the acknowledgement is not granted to the preceding router. If the buffer is not full, the router can accept and acknowledge transferring of packets from the previous router. The desired output port can be obtained from the destination address which is included in the header flit. The routing computation is done by this entity. To do so, the *empty* signal is utilized to enable the output port request signal. If the *empty* signal is inactive (there is a packet in the buffer), the input controller uses the header flit of the packet in the buffer to compute the direction of the packet. If the destination address matches the address of local core, the input controller sends a request to the local port. Otherwise, a request signal is sent to the available output port and waits for grant signal.

- Arbiter: Arbitration is required when more than one input port request the same output port. The arbiter receives the request signals and utilizes a fixed priority scheme to decide which input is granted access to the output port. Input ports have higher priorities than the local port(s).

- Output Status Circuit: The output status circuit keeps track of all output ports to provide information on available output ports. It builds a list of

free output ports where the list is delivered to the input controller entity to quickly decide the available output port.

- Crossbar: The crossbar entity is used to connect the input port with the required output port. According to the selected output port, the crossbar is configured to connect the input port with the desired output port or with the local port. Crossbar entity block diagram and details are shown in Fig. 3.7. It is built using many multiplexors that connect one of input ports to one output port. This stage is composed of five multiplexes to provide all connections possibilities between input and both output and local ports. In the Fig. 3.7 (b), four multiplexes are 3-TO-1 multiplexor to connect both input and local port with the output port and the fifth multiplexor is 5-TO-1 multiplexor. The last multiplexor is used to connect input local port, bidirectional input port1, input port2 with local output port.

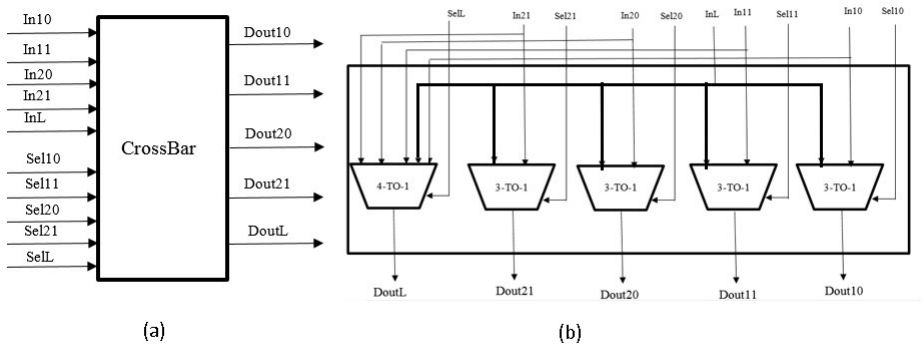


Figure 3.7: Crossbar block diagram.

3.1.3 Multi-local port strategy

The advantages of ring network is being scalable (more than one ring can be implemented). Thus, the packet has higher chance to be forwarded toward its destination than using one ring. The second ring can be used when the first ring is busy which enhances the overall system throughput. On the other hand, as the network size increases, the end to end delay increases and more FPGA resources are consumed. To cope with these problems, a multi-local port mechanism is used where the router can accommodate power of 2 local ports with shared buffer used for all local ports. The round robin is used for selecting among local nodes. In this respect, the multi-local port strategy reduces the end to end delay that a ring suffers from and also reduces resources utilization.

3.1.4 Data Transfer between Routers

The external signals of our router are shown in Fig. 3.8. There are four control signals for each port. Two control signals that are Rqst_out and Ack_out represent the output control signals and the other two signals (Rqst_in and Ack_in) represent the input control. Similarly, two data paths (RD_in for incoming flit and RD_out and outgoing flits). The figure shows two ports: port1 and port2 and each port is considered as a bidirectional port where the data can be received or sent through these ports. Also, the figure shows the transaction of the packets from the first router output port to another router input port and vice versa. The control signals sent by a router to adjacent router to inform the sender router whether its input

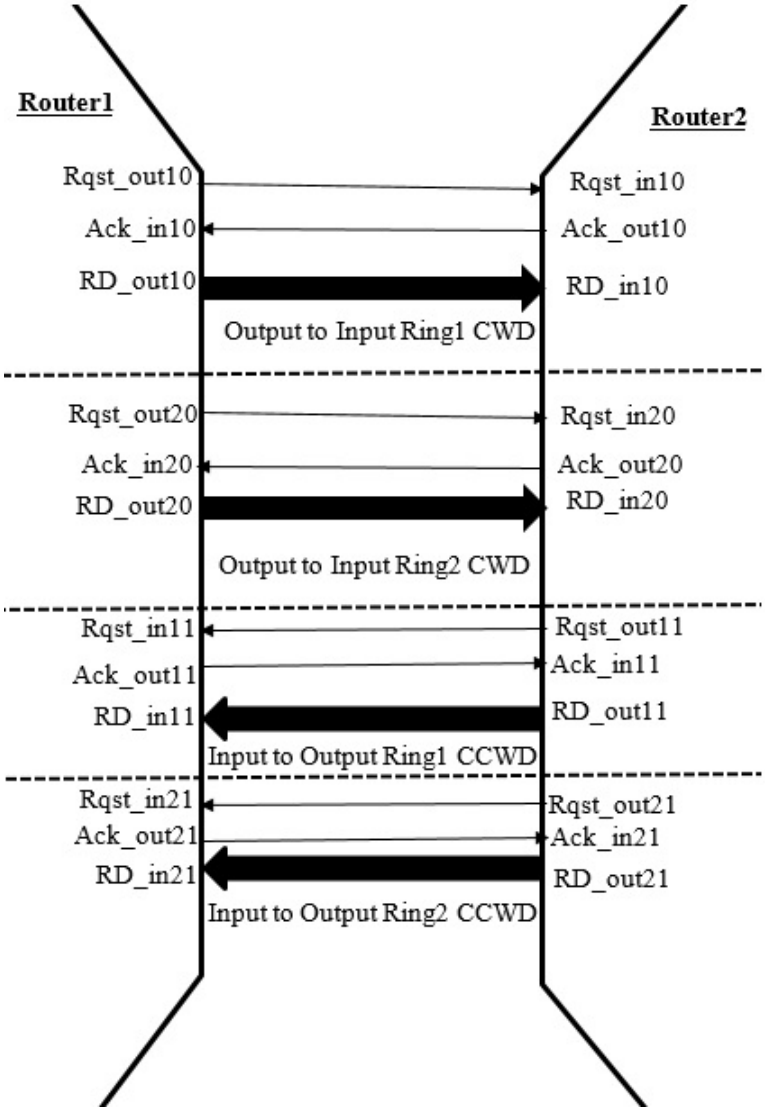


Figure 3.8: Data transfer between adjacent routers.

buffer has enough space to receive the flits or not. To do so, Ack_in of router1 is connected directly with Ack_out of router2. To demonstrate, the sender router sends rqst_out to the receiver router for just only one clock cycle. Then, the receiver router assigns Ack_out to high for one cycle then the transmitted data starts until Ack_out signal becomes low or sender finishes its data.

All the router tasks such as receiving the incoming packets, acknowledgments,

routing and transferring packets are done by the lower level entities of the router. The router entity code only binds these sub-entities together. To provide a better understanding of how they work, assume a local source node injects a header flit into the input port of the FIFO entity. The FIFO entity writes the flit into the buffer. When the flit emerges at the head of FIFO entity, the routing computation reads the address and computes the appropriate path where the input controller sends output port request to the arbiter to perform the required arbitration. When the request is granted, the arbitration result is sent to the configured port of crossbar entity. Then the flow control entity activates read signal which,in turn, leads to injecting the flit to the input port of crossbar entity. The flit then traverses through the crossbar entity from its input port to its output port. Finally, the flit leaves the router as illustrated in the flow char in Fig. 3.9.

Flit Format

The packet is divided into flits where the width of a flit matches the channel width. The first flit includes the routing information that identifies the source and destination address of the packet as well as the packet's injection time stamp. The first two bits are utilized to identify the flit type (header, body, or tail). For the header flit (starts with 10), there are three additional configurable fields (i.e. parameterized in the VHDL code); source address, destination address, and time stamp. Fig. 3.10 shows an example header flit format that supports a 256-core NoC. The destination address is divided into two configurable parts; the first part represents the node address and the second part represents the local port address.

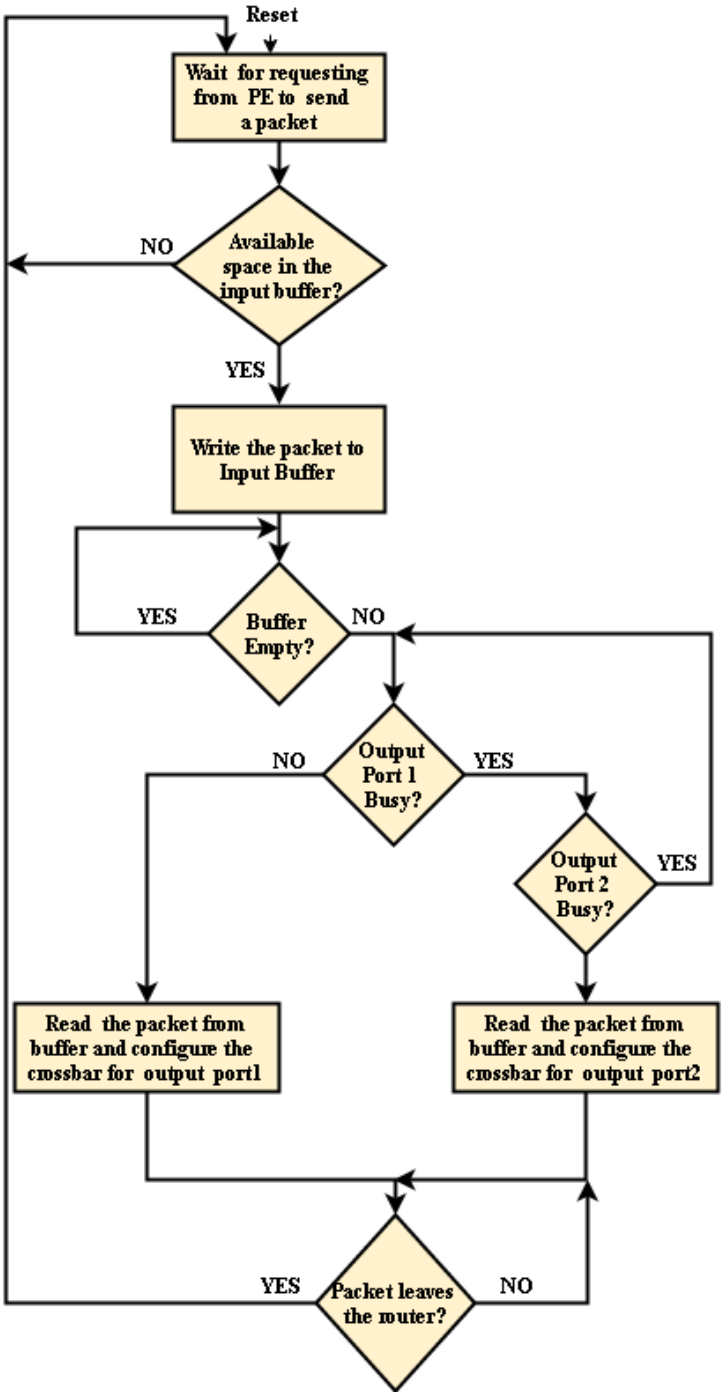


Figure 3.9: Router task flow chart

The adopted addressing supports up to to 16 local ports. Body and tail flits carry the packet's payload that depends on the architecture/application being simulated.

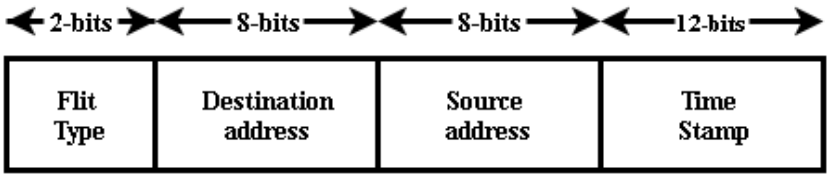


Figure 3.10: Flit structure.

- Header Flit

  First two bits in a flit represent the type of flit and these bits will be equal to "11" in the case of one flit per packet (i.e test packet). The next eight bits represent the destination address followed by eight bits source node address. These eight bits are adequate for 256 nodes in the NoC. Finally, the rest of 12 bits represent the time stamp for the packet. This field is helpful to calculate the packet latency. In our design, the time stamp takes the value of counter when the packet is injected to the network. At the destination node, per packet latency is added to the packets time stamp by the nodes latency model.

CHAPTER 4

IMPLEMENTATION AND ANALYSIS

4.1 Implementation

The FBNoC simulator components have been designed using RTL-VHDL. A 9-core FBNoC (to simulate a 3X3 mesh NoC) has been implemented on a Xilinx Virtex7 XC7VX485T FPGA evaluation board using Xilinx ISE 14.7 tool suite for synthesis and implementation.

Table 4.1 shows the FPGA resource utilization of the FBNoC components for two implementations; a 9-node FBNoC with one local port per node, and a 5-node implementation with two local ports per node. The total implementation time (synthesize, place and route, bit-stream generation) was about 6 minutes on a Core i5 2430m CPU with 2.4GHz processor speed and 8GB of RAM. FBNoC utilizes block SRAMS (BRAMs) to store the simulation traffic traces, user specifications,

Table 4.1: Resource utilization breakdown per component FBNOC on virtex7 XC7VX485T FPGA

	Per Module Resources Utilization			
Module	LUTs	FFs	DSP48E1s	BRAM
Router	924	506	0	0
Traffic Manager	211	75	0	1
Latency Model	216	116	1	0
Regression Model	1812	56	1	1
Total (1 LP per node)	12220	5391	10	11
Total (%1 LP)	4%	1%	0%	1%
Total (2 LPs per node)	9142	3200	6	11
Total (% 2 LP)	3%	1%	0%	1%

and coefficient values of the regression model. The implementation results show that the design consumes 11 BRAMs because nine nodes require nine BRAMS for traffic traces, one BRAM for network configurations, and one BRAM for coefficient values of the congestion regression model (in node 0 only). In addition, the design consumes 10 DSP blocks (DSP48E1s in Virtex7) in case of one local port and 6 DSP blocks for the case of using two local ports. This is because every latency model requires one DSP block (and one additional DSP block for the congestion regression model in node 0). Moreover, the simulator can operate at 300 MHz frequency.

4.2 Measurement Methodology

The popular software based NoC simulator is the Booksim. It is used widely in the NoC research. Therefore, to verify the proposed FBNoC simulator, it is compared against the Booksim. This comparison will provide more confidence in the accuracy of NoC simulator. In addition, the FPGA resource utilization is

obtained by using ISE14.7 tool. To study the effect of multi-local port strategy on resource utilization, we implement three network size 256, 64 and 32 nodes with one local port per node, two local ports per node and four local ports per node. Moreover, a ChipScope tool is utilized to test the design and show how the FBNoC simulator can accurately deliver the packets to their destination for 8 nodes with one and two local ports.

4.3 Design Verification With ChipScope

Traditionally, the debugging and verification of digital circuits is accomplished using Logic Analyzers, connected to a given design via the physical I/O pins of the FPGA being used (this applies equally to situations where ASIC implementation is used). If internal signals need to be monitored, these signals must be assigned to additional I/O pins, which can increase the cost and complexity of the design.

A logic analyzer is a standard digital design debugging tool. It allows the user to view a large selection of signals in a time-correlated fashion, with the signal waveforms captured and presented to the user. Unfortunately, such a useful tool has problems when faced with FPGA debugging, primarily because of the limits of the chip packaging. When debugging an FPGA design, the signals you want to examine are, more often than not, buried inside the FPGA chip and have never been attached to one of the package pins. As the logic analyzer can only connect to the outside of the FPGA package at the pins, it doesnt have access to the internals of your circuit. additional logic can be inserted into the synthesized design. This

logic (ILA and ICON) will collect signal information from your design and store the actual signal values into RAM on the board-under-test. A connection back to the host computer, running the ChipScope tools, allows t to view the signal information. In this section, the FBNoC is tested using ChipScope Pro tool with one local port and multi-local port per node.

4.3.1 Verifying the design with one local port

In this scenario, we implement eight nodes FBNoC network with one local port/node. ChipScope, Fig. 4.1, is utilized to verify the correctness of the design and implementation. As this Figure shows, the shown test packet (i.e consists of one flit) (0x30403FFF) where the first 2 bits of header are 3 (11 in binary). The next 16 bits are the destination and source addresses, 0x04, and 0x03, respectively. The last 12 bits are the time stamp. Thus the packet is sent by Core 03 to the destination Core 04. As this Figure shows, the packet was injected at time 261 (of the simulator clock) and delivered to node 04 at time 276.

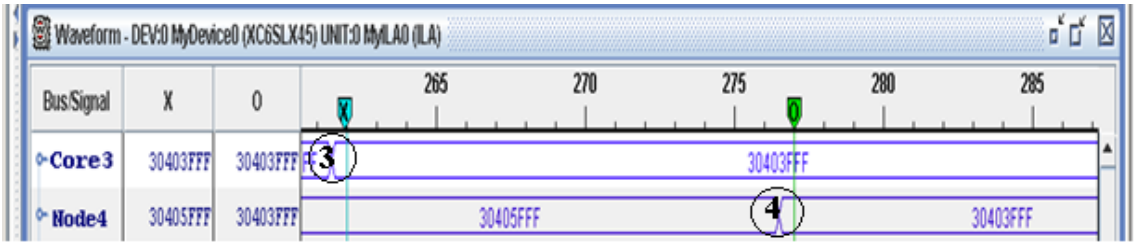


Figure 4.1: ChipScope one local port/node core 03 sends the packet 0x30403FFF to core 04.

4.3.2 Verifying the design with two local ports per Node

In this scenario, two local port per node (i.e each node connects two local cores). ChipScope, Fig. 4.2, is utilized to verify the correctness of the design and implementation. The result of ChipScope shows the full path of test packet (0x30400FFF) from the source Core 00 to the destination Core 04. As Fig. 4.2 shows, the packet was injected at time 261 (of the simulator clock) and received by node 01 at 267. Then the packet traverse the node 01 and received by node 02 at 274 . Finally, the packet is delivered to Core 04 at time 281. In this scenario, the first seven bits of the destination address is utilized to address the node and the least significant bit is utilized to address the local cores connected to the node. For instance, the packet [0x30400FFF] is sent from Core 00 to Core 04 where the destination address is (00000100). Since each node has two local cores connected to it (Core 04 and Core 05 are connected with node 02 in this scenario), the node destination address of the packet is (0000010) that represents the address of node 02 while the least significant bit is the address of Core 04 (i.e 0 for Core 04 and 1 for Core 05) as shown in Fig. 4.2.

4.3.3 Verifying the design with two local ports per node using two flits per packet

In this scenario, two local port per node (i.e each node connects two local cores). ChipScope, Fig. 4.3, is utilized to verify the correctness of the design and implementation. The result of ChipScope shows the full path of test packet

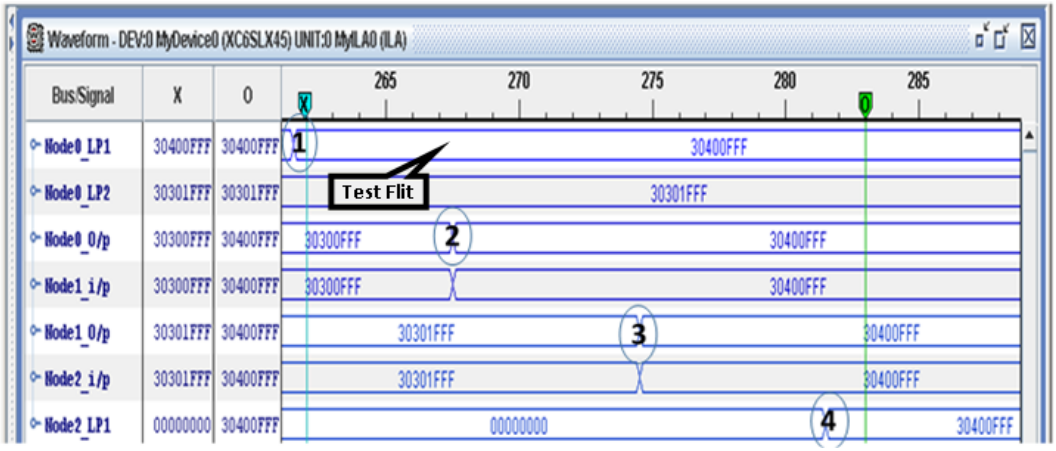


Figure 4.2: Full path ChipScope two local ports Core 03 sends the packet 0x30400FFF to Core 04.

[0x204004F8-0x10504BC4] from the source Core 00 to the destination Core 04. As Fig. 4.3 shows, the packet was injected at time 214 (of the simulator clock) and received by node 01 at 218. Then the packet leaves node 01 and arrives node 02 at 225. Finally, the packet is delivered to Core 04 at time 234. Like the previous scenario, the first seven bits of the destination address is utilized to address the node and the least significant bit is utilized to address the local cores connected to the node. For instance, the packet [0x204004F8-0x10504BC4] is sent by Core 00 to Core 04 where the destination address is (00000100). Since each node has two local cores connected to it (Core4 and Core5 are connected with node 02 in this scenario), the node destination address of the packet is (0000010) that represents the address of node 02 while the least significant bit is the address of Core 04 as shown in Fig. 4.3.

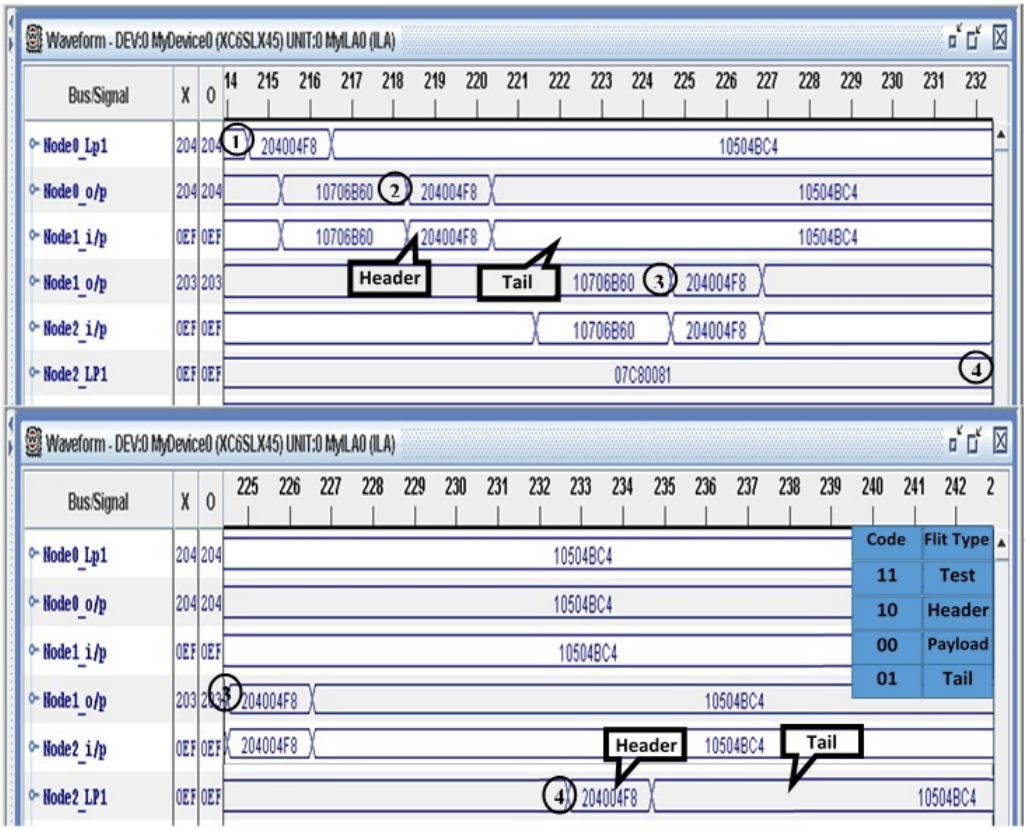


Figure 4.3: Full path ChipScope for two local ports per node of packet [0x204004F8-0x10504BC4] sent by Core 03 to Core 04.

4.4 The Effect of Using A multi-local Port on FPGA Resource Utilization And Simulation Speed

To evaluate the effect of using multiple local ports per NoC node on resource utilization and simulation performance, three NoC sizes (256, 64, and 32 cores) have being synthesized and simulated using a synthetic trace consisting of 100,000 packets with uniform distribution. Each NoC was designed and simulated with 1, 2, 4, and 8 LPs/node, using 3 different buffer sizes (in terms of number of

packets); 4, 8, and 16 packets. Hence a total of 36 design points were evaluated at injection rate of 0.05 packet/node/cycle. Furthermore, resources are reported as FFs and LUTs, separately. Simulation speed (packets/cycle) is measured as the total number of packets injected and received over the total simulation time (number of FBNoC cycles from the injection of 1st packet till the reception of the last packet). Both simulation speed and resource utilization have been normalized to the speed with 1 LP/node and the resource utilization with 8 LPs/node, respectively as illustrated in the Fig. 4.4.

4.5 ANALYSIS

In this section, The FBNoC accuracy is verified under synthetic and realistic traffic.

4.6 FBNoC Accuracy Evaluation

To verify the accuracy of the proposed model, we simulate two types of traffics that are synthetic traffic and realistic traffic. The FBNoC can achieve good accuracy with 0.95 confidence interval.

4.6.1 Verification With Synthetic Traffic

In this section, we use the Booksim, the popular software based NoC simulator, as a reference to validate FBNOCs accuracy. Booksims accuracy is widely

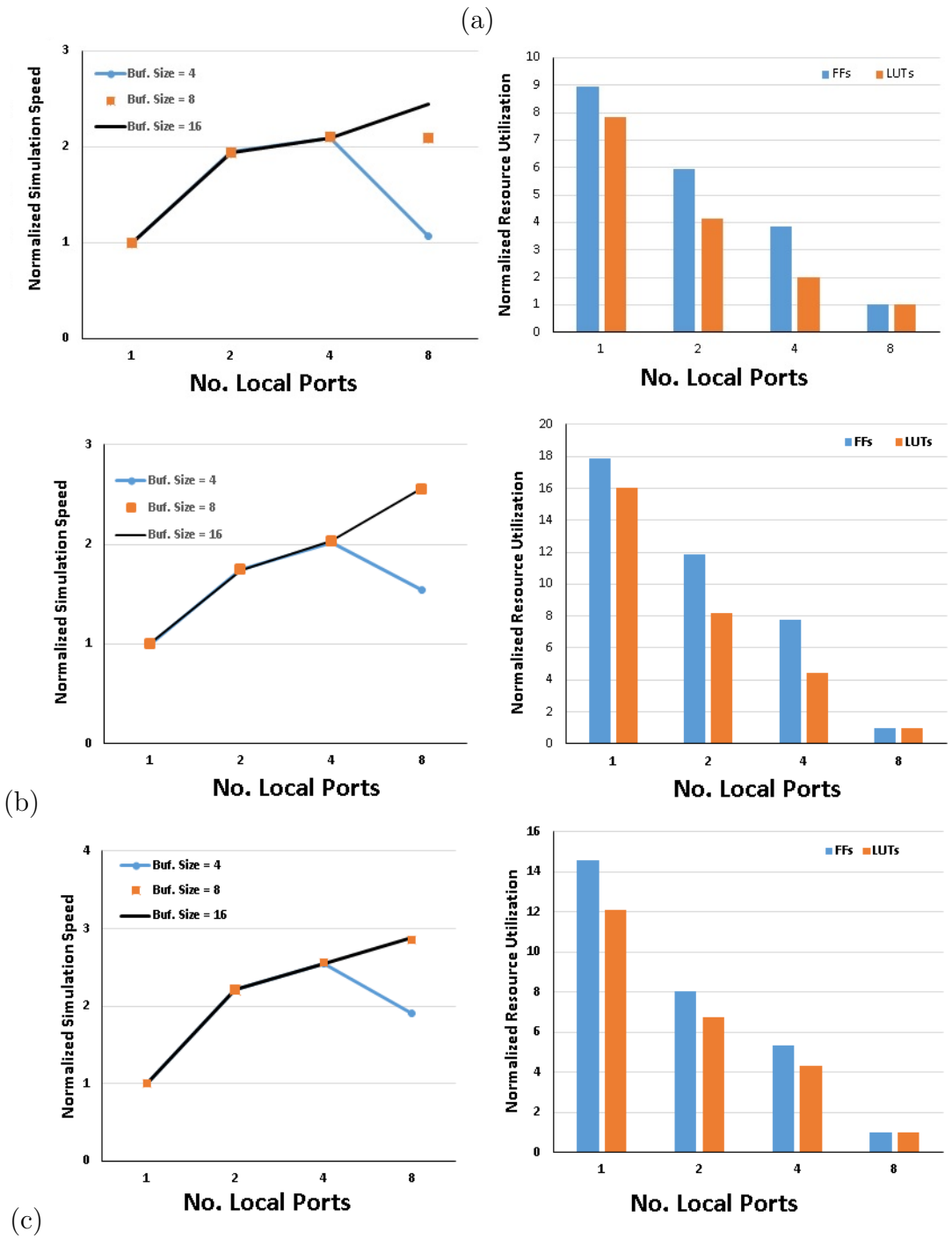


Figure 4.4: Normalized Resource utilization and simulation speed for three network sizes versus the number of local ports per node for several sizes of the buffer shared between LPs. (a) 32 Cores NoC size., (b) 64 Cores NoC size., (c) 256 Cores NoC size.

accepted among NoC researchers and developers. It has three phases which are warm-up, measurement, and drain phases. The warm-up phase is used to bring the network into steady state where the generated packets are not counted in this phase while the measurement phase comes after the warm-up phase and the generated packets during this phase are called measurement packets. Finally, the drain phase represents the required time for measurement packets to reach their destinations. In addition, the drain phase may never finish when the network start starvation. Hence, the simulation is terminated when the latency exceeded the threshold value as a solution for infinite latency drain. To verify the accuracy of the proposed models in FBNoC, two NoC topologies were simulated using synthetic traffic traces with uniform address distribution; mesh and fat tree topologies for three network sizes (9, 64, and 256 nodes) for the mesh and (8, 64, and 256 nodes) for the fat tree. The NoC configurations of both networks are shown in Table 4.2. It should be noted that though the FBNoC itself uses simple ACK-NACK flow control, the modeled NoCs use the popular credit-based flow control scheme (i.e. the latency and congestion regression models were obtained with this flow control latency data). We simulated 15000 warm-up cycles, 15000 measurement cycles, and drain phase. Fig.11 shows the average packet latency versus injection rate curves for 3x3, 8x8 and 16x16 mesh and 8, 64, 256 nodes fat tree with 2,4, and 6 virtual channels for each network.

Fig.4.5 (a) to (c) show the average packet latency of 3x3 mesh for variety of injection rate. The figures explain the effect of increasing the injection rate and

Table 4.2: Configuration parameters for the two NoCs used for evaluating the accuracy of FBNoC against that of Booksim.

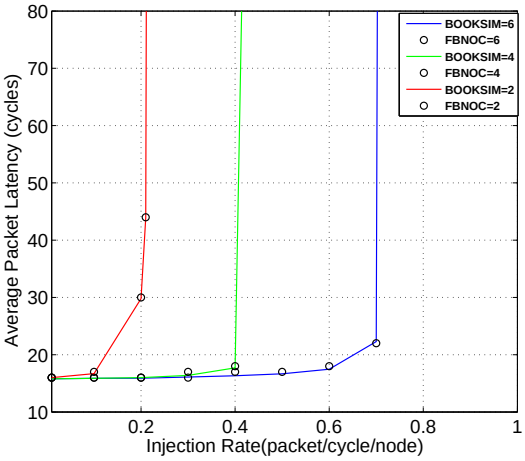
Parameter	Mesh	Fat tree
Topology	Mesh	Fat tree
Number of Nodes	9,64,256	8,64,256
VC Buf. size	16 flits	16 flits
Flow control	Credit-based	Credit-based
VC Allocator	Islip	Islip
Sw allocator	Islip	Islip
Routing delay	1	1
Routing algorithm	DOR	NCA
VC Alloc delay	1	1
Num VCs	2, 4,6	2, 4,6
Traffic	uniform	uniform
Packet-size	1,4,5 Flits	1,4,5 Flits

the packet size on average packet latency. It can be seen that as the packet size increases, the network reaches to the saturation point earlier. This behavior is shown in the Fig. 4.5 (a) to (c).

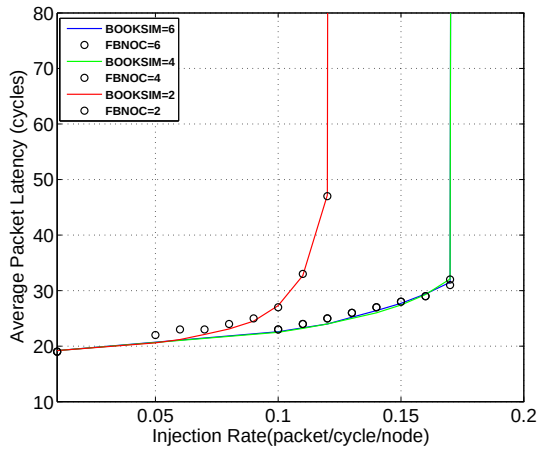
Fig. 4.6 (a) to (c) and Fig. 4.7 (a) to (c) show the average packet latency of 8x8 mesh and 16x16 mesh respectively for variety of injection rate.

Fig. 4.8 to Fig. 4.10 show the average packet latency curves of fat tree for 8, 64 and 256 nodes for virtual channel equals to 2 ,4, and 6 for each network.

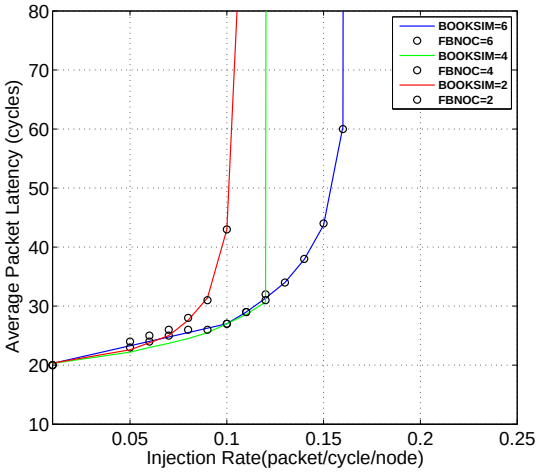
It can be noted that the average packet latency can be reduced by dividing the physical channel into several virtual channels. In addition, at higher injection rates, the number of virtual channels becomes more important because the network saturates at higher rates. This behavior is exhibited in the Fig. 4.5 to Fig.4.10. For example, in Fig. 4.7, the network saturates at injection rate of



(a)

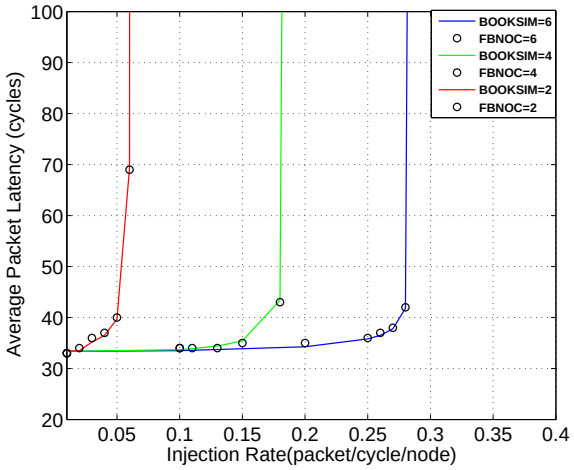


(b).

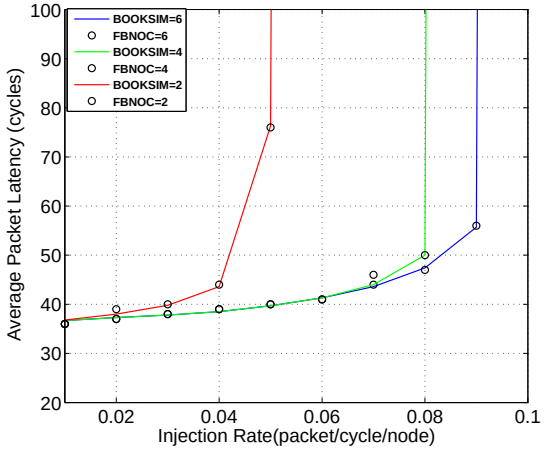


(c).

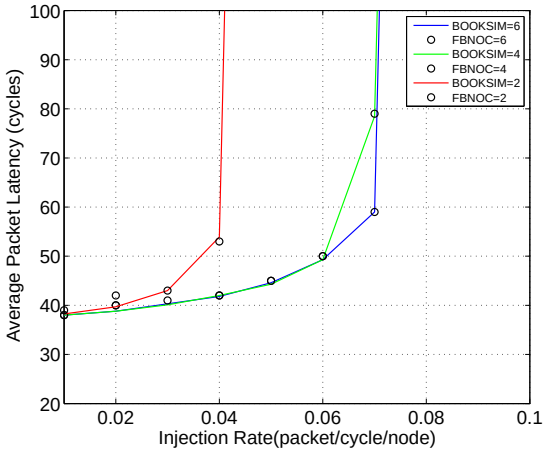
Figure 4.5: Average packet latency with (2,4,and 6 VCs) for 3x3 mesh topology (a) packet size = 1 flit, (b) packet size = 4 flits , and (c) packet size = 5 flits. proposal versus BookSim.



(a)

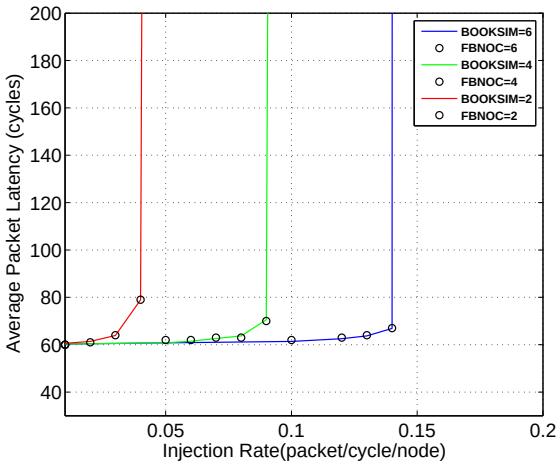


(b).

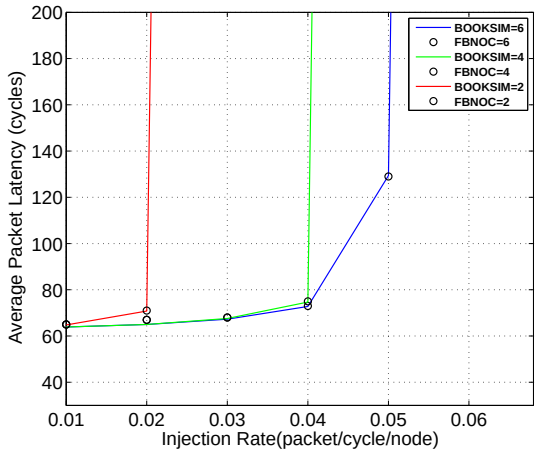


(c)

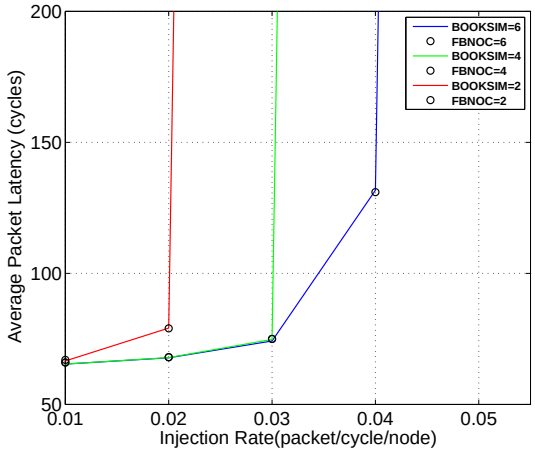
Figure 4.6: Average packet latency with (2,4,and 6 VCs) for 8x8 mesh topology (a) packet size = 1 flit, (b) packet size = 4 flits , and (c) packet size = 5 flits. proposal versus BookSim.



(a)

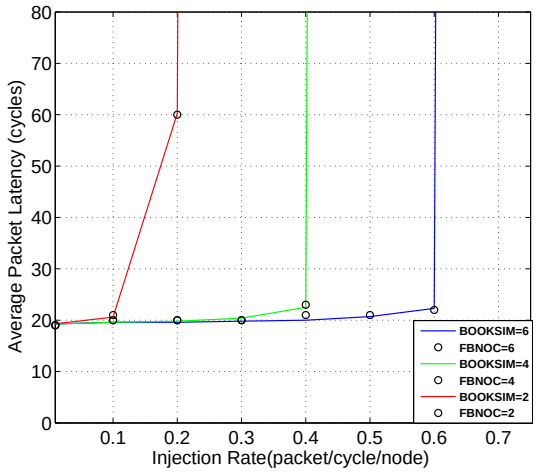


(b).

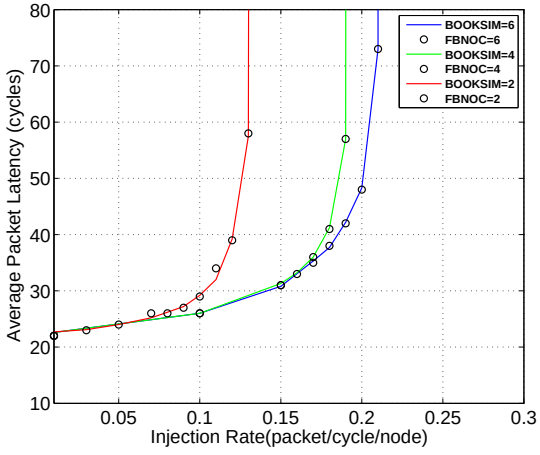


(c).

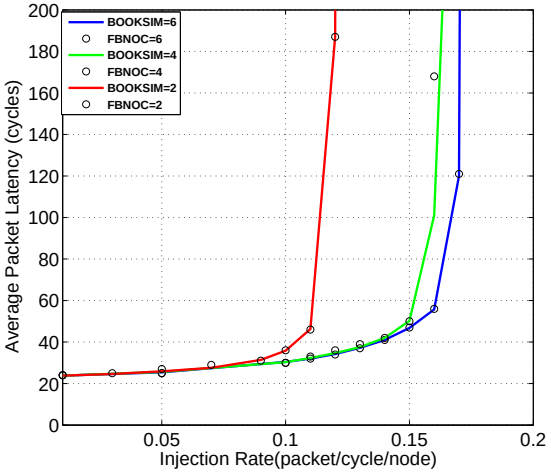
Figure 4.7: Average packet latency with (2,4,and 6 VCs) for 16x16 mesh topology (a) packet size = 1 flit, (b) packet size = 4 flits , and (c) packet size = 5 flits. proposal versus Booksim.



(a)

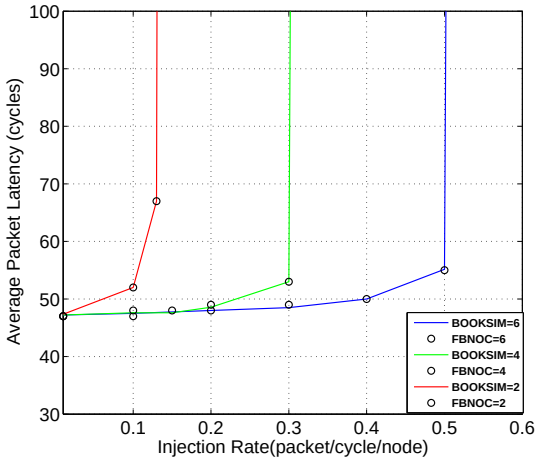


(b).

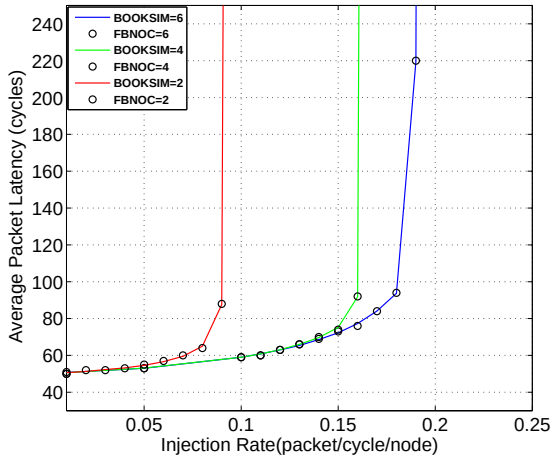


(c).

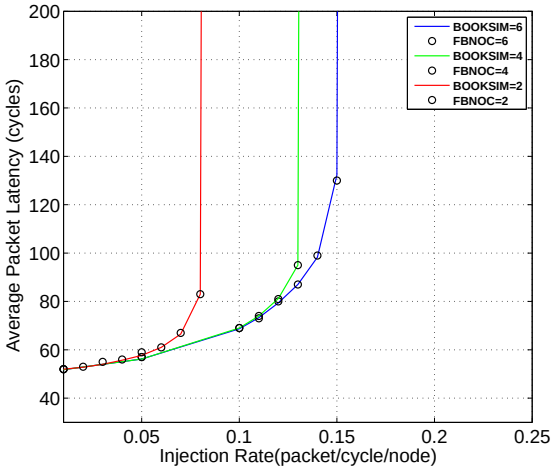
Figure 4.8: Average packet latency with (2,4,and 6 VCs)for 8 nodes fat-tree topology (a) packet size = 1 flit, (b) packet size = 4 flits , and (c) packet size = 5 flits. proposal versus Booksim.



(a)

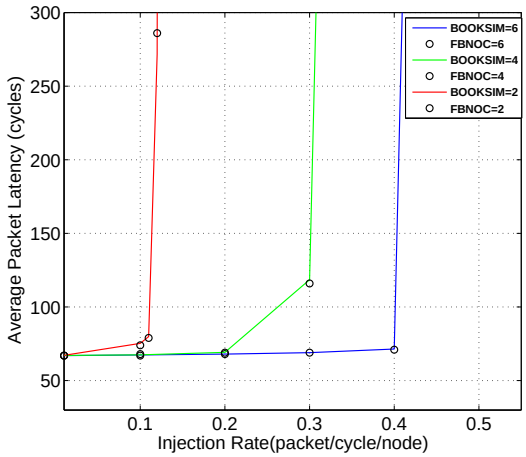


(b).

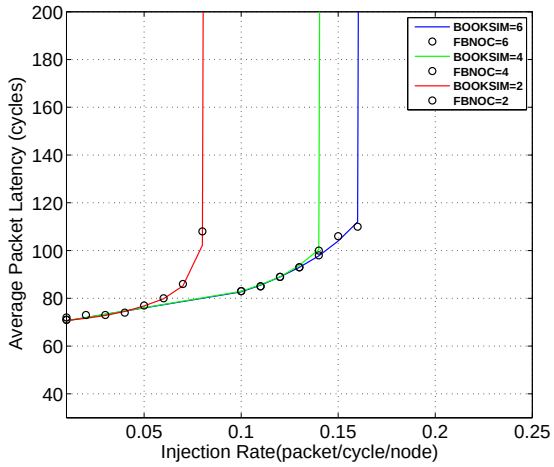


(c).

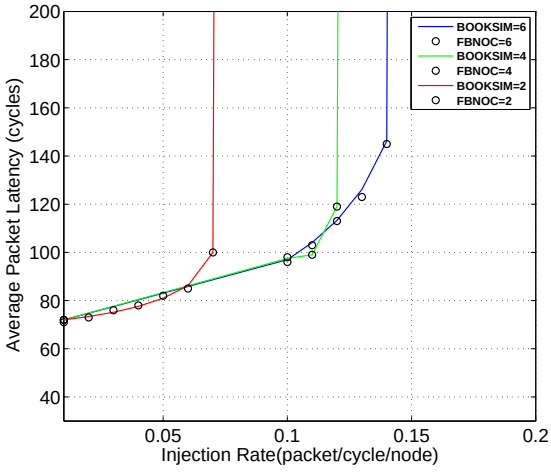
Figure 4.9: Average packet latency with (2,4,and 6 VCs)for 64 nodes fat-tree topology (a) packet size = 1 flit, (b) packet size = 4 flits , and (c) packet size = 5 flits. proposal versus Booksim.



(a)



(b).



(c).

Figure 4.10: Average packet latency with (2,4,and 6 VCs)for 256 nodes fat-tree topology (a) packet size = 1 flit, (b) packet size = 4 flits , and (c) packet size = 5 flits. proposal versus Booksim.

0.04 packet/cycle/node when VCs = 2, 0.09 packet/cycle/node when VCs = 4, and 0.14 packet/cycle/node when VCs = 6. The figures also show the effect of network size on average packet latency. It can be seen that as the network size increases, the average packet latency increases and the network reaches to the saturated point earlier than small network size. This is because the end to end delay (number of hops) increases as the network size increases and more packets are sent through the network. We can see that the FBNoC simulator curves almost have the similar packet latency with the Booksim. There are a slight differences between the two curves due to the regression error in the latency model.

4.6.2 Verification With Real Traffic Traces

Since one of the main design objectives of FBNoC was to be able to deliver actual traffic when integrated into an architectural simulator, it becomes incumbent to verify its accuracy and performance in such situation, i.e. with real traffic traces. Real traffic traces differs significantly from synthetic traffic in terms of traffic burstness and address distribution. Hence, four benchmark applications were used for this evaluation; FPPPP, Sparse, RS_32_ 28_ 8 dec, and ROBOT. A prominent multi-core simulator, GEM5 [36] was used to generate the communication traffic for these applications on a 4X4 Mesh NoC (i.e. with 16 cores). Gem5 is a cycle-accurate full-system multi-core simulator. Garnet [2] is a cycle-accurate interconnect simulator that is built-in within GEM5. Each of the simulated applications consists of several tasks with temporal dependencies and

Table 4.3: **Communication Characteristics of the Applications Used to Generate Real Traffic Traces.**

Benchmark	FPPPP	Sparse	ROBOT	RS_32_28 _8 dec
Avg Burst Size	50.7	290	50	∼2
Avg Inter-Burst-time	39	623	397	8
Average NoH	2.24	1.72	2	2.16

inter-task communications. Each task is mapped to a core with communication-driven assignment (i.e. communicating tasks are placed close to one another whenever it is possible). Table 4.3 shows the communication characteristics of the four applications used to generate the realistic traffic traces in terms of average traffic burst size, average inter-burst time (defined as the average time between consecutive bursts), and average number of hops per packet (NoH). As the Table 4.3 and Fig. 4.11 show, the selected applications exhibits different communication patterns in terms of burst size and inter-burst time, while the average NoH is around 2 due to the optimum task assignments to the cores. GEM5 was configured to model a 16 cores SoC where it reads the benchmarks from a disk image and to record all communication traces consisting of all packets that enter and exit the network with the respective time stamps (source and destination addresses, and injection and reception time stamps). These traces were then used with FBNoC where each packet was injected into the NoC at the source address at the specified time. Packets latencies calculated by the destination nodes in FBNoC where then compared to the latencies obtained from the GEM5 traces, Fig. 4.12. As this figure shows, the packet latencies obtained from FBNoC matches those obtained from GEM5 very well.

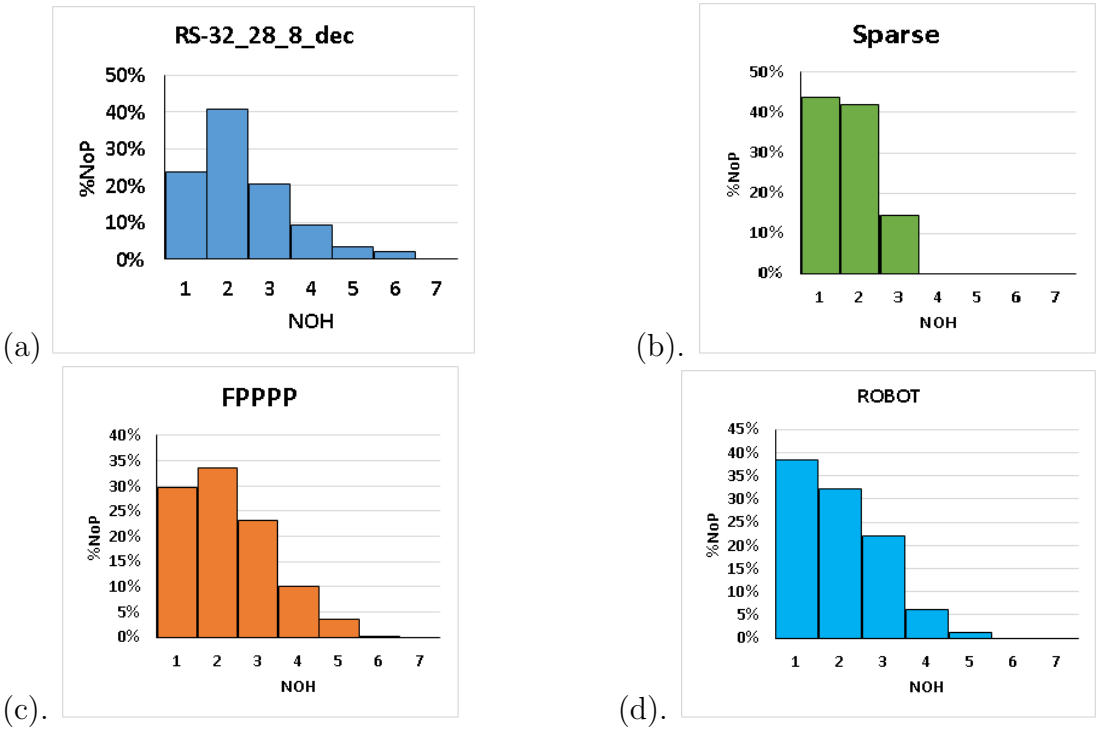


Figure 4.11: Number of Hops (NOH) Histogram for the 4x4 mesh for the four benchmark traces.

4.7 Performance Evaluation

In this section, the speedup of FBNoC is compared against Booksim. In addition, the FPGA resource utilization is compared with DART and FIST simulators.

4.7.1 Simulation Speed

We measured the simulation time of Booksim on PC Core i5 2430m CPU with 2.4GHz processor speed and 8GB of RAM. The simulation speed of FBNoC was compared to that of Booksim using synthetic traffic for an 8x8 mesh NoC described in table 4.2. Using synthetic traffic allows stressing out the NoCs (i.e. inducing more congestion on demand) and evaluating the simulators performance under such conditions. Fig.4.13 shows FBNoC speedup over Booksim (i.e. Book-

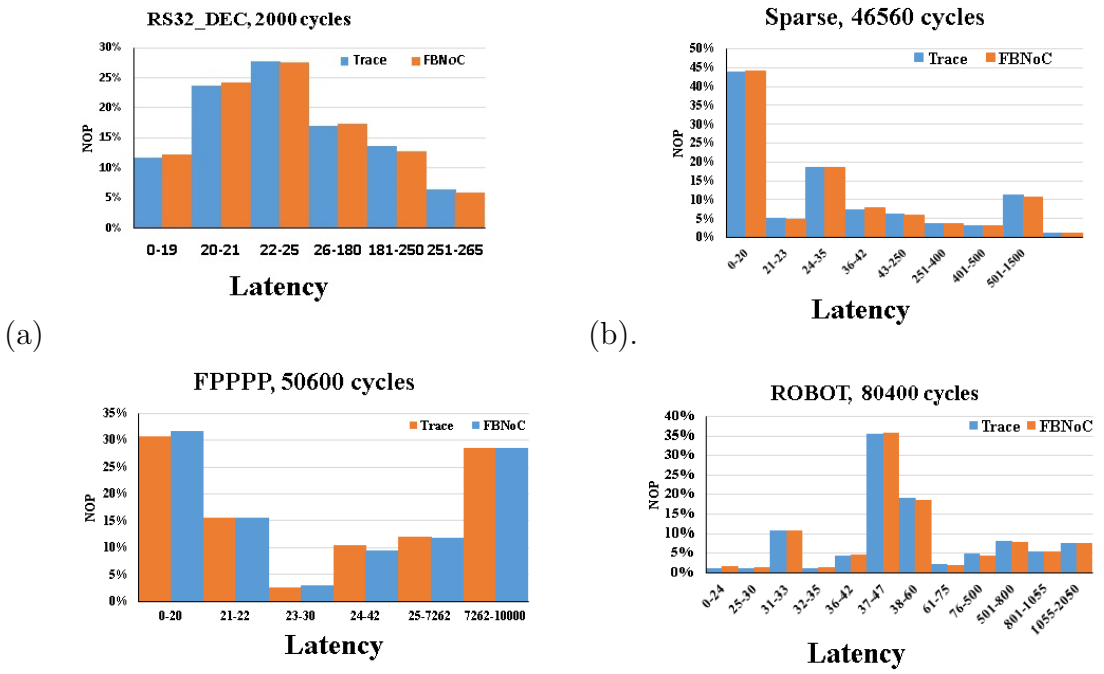


Figure 4.12: Packet Latencies Distribution obtained from GEM5 traces and FBNoC simulations for a 4x4 mesh with four different benchmarks: (a) RS-32 28 8 dec., (b) Sparse., (c) FPPPP., and (d) Robot. : proposal versus Trace.

sim simulation time/FBNoCs) versus the injection rate. The simulation time of Booksim increases as the injection rate increases while FBNoCs remains almost constant. Hence, the higher the injection rate, the greater the speedup (e.g. the speedup at the low injection rate of 0.01 is 5,000X and jumps to over 20,000X at 0.04 injection rate). Moreover, as the network size increases, the speedup gap expands further. Though DART [10] has also reported high speedup over Booksim (up to 100X), resource contention were not implemented accurately because DART uses a single-cycle router with one output port. Thus, at higher injection rates, its accuracy diverge from that of Booksim. Still, FBNoC speedup is orders of magnitude higher that those of DART with excellent accuracy matching with

Booksim over all injection rates for different NoCs as was shown in Fig. 4.13.

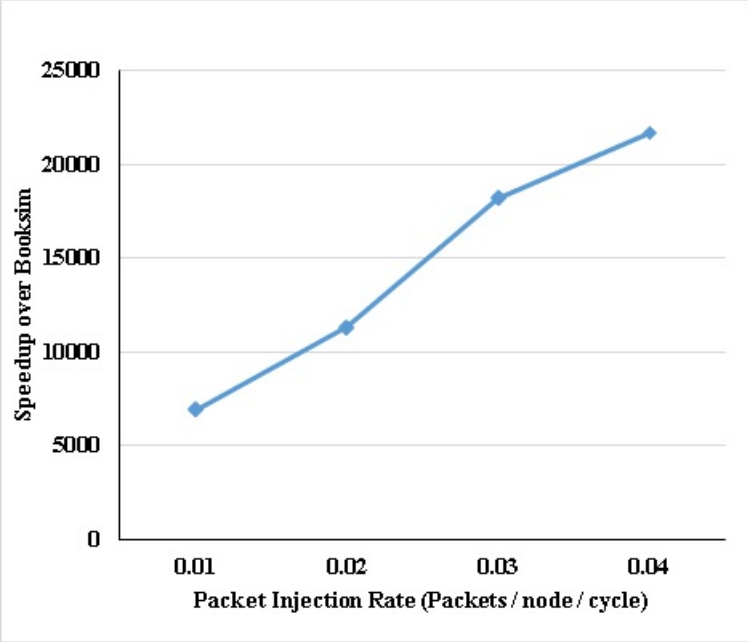


Figure 4.13: Speedup of FBNoC over Booksim when simulating an 8x8 mesh.

4.7.2 Resources utilization vs DART and FIST

Table 4.4 shows a comparison between FBNoC and DART [10]. in terms of FPGA resources utilization. The results show that when using 2 LPs per node, DART consumes almost 2.8 times LUTs and 4.1 time FFs more than FB-NoC. When FBNoC is configured with 1LP per node, DART consumes almost 2.1 times LUTs and 2.4 time FFs. In all cases, Dart consumes more resources (LUTs,FFs,BRAMs) than FBNoC. In contrast, the FBNoC consumes 6 and 10 DSP48E1s in 2LP and 1LP respectively due to the latency model.

Table 4.5 shows a comparison between FBNoCs latency model with that of FISTs in terms of resource utilization and maximum frequency for several NoC

Table 4.4: FPGA resources utilization for 9-nodes FBNoC and DART.

Simulator	LUTs	%LUTs	FFs	%FFs	BRAMs	%BRAMs	DSP 48E1s
FBNoC(1 LP per node)	12220	4%	5391	1%	11	1%	10
FBNoC(2 LPs per node)	9142	3%	3200	1%	11	1%	6
DART	26380	9%	13192	2%	99	10%	0

sizes. FBNoC was synthesized using the same FPGA reported in [12] for FIST. It should be noted that FISTs latency model would have to be replicated in all network nodes and uses BRAMs extensively to store the load-delay curves. Hence, as the network size increases, the required number of BRAMs increases which consequently limits FIST scalability. In contrast, in our proposed latency model, the congestion module is only included in Node 0 and the ACPR value is forwarded to all nodes at regular intervals using additional parallel ring. Hence, it requires one BRAM to store the coefficient of regression model and fewer LUTs than FIST. In addition, our proposed latency model can run at higher frequency than FIST.

Table 4.5: FPGA resource utilization (BRAMs, LUTs) and frequency for FIST and FBNoC.

	FBNOC:Virtex6 LX760				FIST :Virtex6 LX760		
Network Size	Block RAMs	LUTs	%LUTs	Freq. MHz	Block RAM	%LUTs	Freq. MHz
4x4	1	1135	0%	582	8	0%	448
8x8	1	5538	1%	578	32	1%	443
12x12	1	10237	2%	557	72	2%	375
16x16	1	18035	4%	554	129	5%	375

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this chapter, we conclude the thesis work presented in the previous four chapters including introduction, literature review, effect of using multi-local port strategy on FPGA resources utilization, and the novel idea of using of a latency model.

5.1 Conclusion

A resource-efficient FPGA-based NoC simulator (FBNoC) that can be integrated with FPGA-based manycore architectural simulators is introduced. On bar with the best SW NoC simulators accuracy, it is extremely faster (by at least three orders of magnitude). Compared to other FPGA-based NoC simulators, it achieves better accuracy, speed and uses less FPGA resources. Its ability to be used as a stand-alone NoC simulator with actual traffic traces or synthetic traffic has been demonistrated. FBNoC can model and simulate several popular NoC topologies

accurately and efficiently without the need to re-synthesize for different NoCs. Results show that FBNoC can achieve more than 20000X speedup over the popular SW NoC simulator Booksim.

5.2 Contributions

This thesis makes the following contributions:

- Developed a hardware FPGA-based NoC simulator (FBNoC) that can model and simulate some NoC accurately and efficiently without code modification.
- The multi-local port strategy with two bidirectional rings network is utilized to decrease the FPGA resources utilization and end to end delay.
- Developed a multi-variable regression latency model that can accurately and efficiently calculate the latency per packet for the simulated network depending on the network size, traffic injection rates, and number of virtual channels per router.
- The FBNoC can cooperate with other hardware simulators up to 256 cores.

5.3 Future Work

This work can be extended to include more topologies. In addition, since the congestion per router is estimated by the latency model, the simulator can be extended to include adaptive routing protocols.

FBNoC regression models can be extended to work with 3D NoCs. Different

regression models with two different types of Hops (horizontal and vertical) required to be developed.

APPENDIX

A. FBNoC User Manual

This document describes the use of the FBNoC interconnection network simulator.

Configuration Input File

The user can configure the parameters through an input file included in the FBNoC directory. The input file is "IN_File and it contains configuration information for the simulator. So, for example, to simulate the performance of a simple 3x3 mesh network on uniform traffic, a configuration such as the one shown in Figure 1 could be used. This simple example uses mesh topology

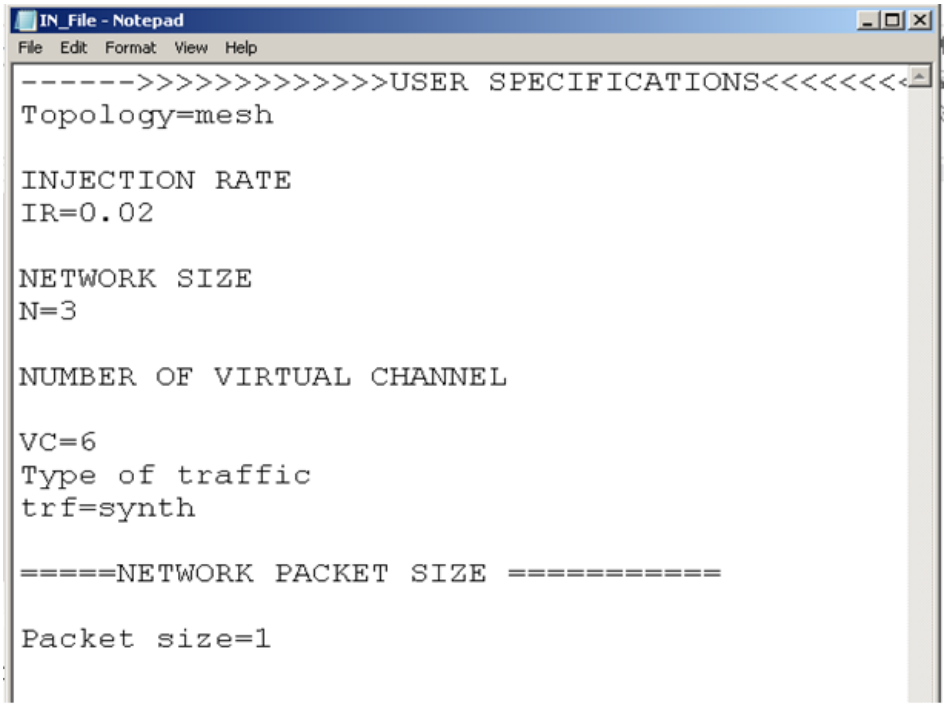


Figure 5.1: Input File Configuration.

with six virtual channels. The injection rate parameter is added to tell the simulator to inject (on average) 0.02 packets per simulation cycle per node. Packet size defaults to a single flit. Any parameters not specified by the user will take on default values. The default value for every parameter in the simulator is specified in the entity FBNoC traffic manager. These parameters for latency model calculation requirement. However, the input configurations are loaded to BRAM in hexadecimal form. Thus, they should be written in hexadecimal format as shown in the figure below.

In this case

- Topology (1st line) is 0 (Mesh), 1 (Fattree) , 2 (Torus), 3 (Ring). Injection rate (2nd line) is written in percentage. For example, 0.05 is written 5.
- Number of virtual channels (3rd line).
- Type of traffic (4th line) 0 (Synthetic), 1 (Realistic).
- Packet size (5th line).

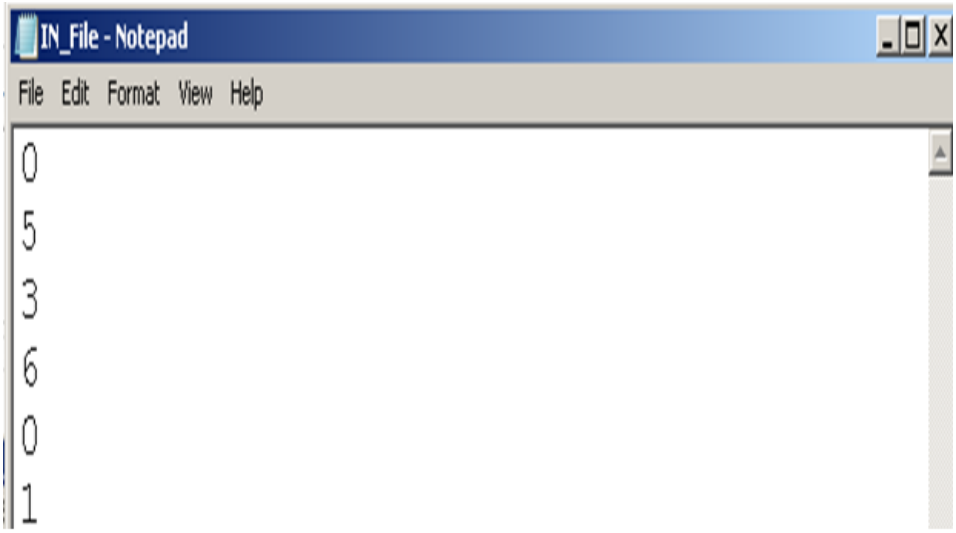


Figure 5.2: Input File Configuration in Hexadecimal Format.

Simulator Network Configuration

For the simulator network configurable parameters, it should be noted that the parameters that require re-synthesis such as network size, buffer size, and number of local ports for the simulator network should be constant and generic. Thus, the user can not specify them from the input file.

To configure the simulator network, the user can use the generic package called conj (the package contains generic parameters for the simulator network) as shown in figure

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

PACKAGE conj IS
constant net_size:integer:=4;
constant LP:integer:=1;
constant wdth: integer:=30;
constant B_len: integer:=4;
       type Mdata is array (integer range <>) of std_logic_vector(17 downto 0);
       type cnj is array (integer range <>) of std_logic_vector(Lp-1 downto 0)  ;
       type DmS is array(net_size-1 downto 0)of integer range 0 to Lp-1  ;
       type gmm  is array(natural range<>)of std_logic_vector(29 downto 0);
       type data1 is array(natural range <>)  of gmm(LP-1 downto 0);
       type data2 is array(natural range <>) of gmm(1 downto 0);
       type data4 is array(natural range <>) of gmm(3 downto 0);
       type data16 is array(natural range <>) of gmm(15 downto 0);
       type data8 is array(natural range <>) of gmm(7 downto 0);
       type nohp  is  array(natural range <>) of integer range 0 to 32;
       type ltncy  is  array(natural range <>) of integer range 0 to 10000;
       type srcc is array(integer range<>)of integer range 0 to 256;
END;
```

Figure 5.3: Generic Parameters for Simulator Network.

The simulator can be used as stand-alone with synthetic traffic or real traffic trace. In both cases, the traffic should be stored in the traffic files called initialFile as seen in the figure bellow. The figure shows 16

initialFile files (i.e. traffic for 16 cores). Therefore, the number of files should be equal to the network size. When the simulator is run, the traffic is loaded to the memories. The traffic manager will responsible to inject the packets to the network and ejecting them from the network.

Figure 5.4: Traffic File for 16 Cores.

The main difference between stand-alone real trace and synthetic traffic (synth) is that in the real traffic, the injection rate is estimated by the network while in the synthetic traffic, the injection rate used is the user specifies.

How to use FBNoC with Full System Simulator

To use FBNoC with full system simulator, the traffic manager entity should not be used. Instead, the NoC module becomes the top level

module in FBNoC and the signals used are DinL: is an input signal (array of network size input signals that are connected to the cores) and each one is with corresponding LPs. For example

* DinL(0) :Node 0 input port signals.
* DinL(0)(0) represents Node 0 LP1 (Core 0)
* DinL(0)(1)represents Node 0 LP2 (Core 1)
* DinL(0)(2) represents Node 0 LP3 (Core 2) and so on.

DotL : is output signal (array of network size input signals that are connected to the cores) and each one is with corresponding LPs :

* DotL (0) : Node 0 output port signals.
* DotL (0)(0) represents Node 0 LP1 (Core 0).
* DotL (0)(1) represents Node 0 LP2 (Core 1).
* DotL (0)(2) represents Node 0 LP3 (Core 2).

B. Using GEM5 to Generate Realistic Traffic

– Downloading gem5

git clone http://gem5.googlesourse.com/public/gem5.

– Setup the following Files

sudo apt-get install g++

sudo apt-get install python

sudo apt-get install python-dev

sudo apt-get install swig

sudo apt-get install zlib

sudo apt-get install m4

- Create Disk image

  ./util/gem5image.py init Reldtc-trfc(image name) 4096(memory size)

- Downloading Benchmark as image disk and save it in disk image. if you have not mounted the disk image, do so now, make sure to unpack the core fs to this disk image before continuing mount -oloop,offset=32256 /tmp/Ubuntu-arm.img /mnt cd /mnt mount -o bind /proc /mnt/proc mount -o bind /dev /mnt/dev mount -o bind /sys /mnt/sys cp /etc/resolv.conf /mnt/etc/ chroot .

  // enable the universe repo in etc/apt/sources.list apt-get update apt-get install ubuntu-minimal apt-get install build-essential apt-get install vim apt-get install gcc-multilib apt-get install g++-multilib apt-get install ¡what ever other package

  mkdir mnt ../../util/gem5img.py mount Reldtc-trfc.img mnt wget http://cdimage.ubuntu.com/ubuntu-core/releases/14.04/release/ubuntu-core-14.04-core-amd64.tar.gz

  /gem5 build/ARM/Gem5.opt configs/example/fs.py disk-image /home/bpayne/full-system-image/disk/arm-ubuntu.matty- Reldtc-trfc.img

- Invoke garnet network in gem5

  ./build/ALPHA/gem5.debug

configs/example/ruby_random_test.py

–num-cpus=16

–num-dirs=16

–network=garnet2.0

–topology=Mesh_XY

-mesh-rows=4

Flit type	Destination address	Source address	Time stamp
3	6	1	2F6
3	6	1	302
3	5	1	336
3	9	1	36A
3	3	1	323
3	2	1	450
3	2	1	48B
3	2	1	4C2
3	3	1	4FA
3	9	1	535
3	4	1	56C
3	6	1	5A8
3	5	1	51F
3	0	1	61D
3	2	1	653
3	7	1	688
3	6	1	456
3	6	1	490
3	5	1	4C6

The above table shows sample traffic used by FBNoC.

REFERENCES

[1] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks.* Elsevier, 2004.

[2] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on.* IEEE, 2009, pp. 33–42.

[3] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, "Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 4, pp. 31–34, 2004.

[4] V. Puente, J. A. Gregorio, and R. Beivide, "Sicosys: an integrated framework for studying interconnection network performance in multiprocessor systems," in *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on.* IEEE, 2002, pp. 15–22.

[5] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J.-H. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on.* IEEE, 2013, pp. 86–96.

[6] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, and F. Catthoor, "A complete network-on-chip emulation framework," in *Design, Automation and Test in Europe, 2005. Proceedings.* IEEE, 2005, pp. 246–251.

[7] Y. E. Krasteva, F. Criado, E. de la Torre, and T. Riesgo, "A fast emulation-based noc prototyping framework," in *Reconfigurable Computing and FPGAs, 2008. ReConFig'08. International Conference on.* IEEE, 2008, pp. 211–216.

[8] G. Schelle and D. Grunwald, "Onchip interconnect exploration for multicore processors utilizing fpgas," in *2nd Workshop on Architecture Research using FPGA Platforms*, 2006.

[9] P. T. Wolkotte, P. K. Hölzenspies, and G. J. Smit, "Fast, accurate and detailed noc simulations," in *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on.* IEEE, 2007, pp. 323–332.

[10] D. Wang, C. Lo, J. Vasiljevic, N. E. Jerger, and J. G. Steffan, "Dart: a programmable architecture for noc simulation on fpgas," *Computers, IEEE Transactions on*, vol. 63, no. 3, pp. 664–678, 2014.

[11] T. Van Chu, "Ultra-fast and accurate simulation for large-scale many-core processors," Ph.D. dissertation, Tokyo Institute of Technology, 2015.

[12] M. K. Papamichael, J. C. Hoe, and O. Mutlu, "Fist: A fast, lightweight, fpga-friendly packet latency estimator for noc modeling in full-system simulations," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*. IEEE, 2011, pp. 137–144.

[13] A. B. Nejad, M. E. Martinez, and K. Goossens, "An fpga bridge preserving traffic quality of service for on-chip network-based systems," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.

[14] M. K. Papamichael, "Fast scalable fpga-based network-on-chip simulation models," in *Formal Methods and Models for Codesign (MEMOCODE), 2011 9th IEEE/ACM International Conference on*. IEEE, 2011, pp. 77–82.

[15] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip," in *Computers and Digital Techniques, IEE Proceedings-*, vol. 150, no. 5. IET, 2003, pp. 294–302.

[16] E. Rijpkema, K. Goossens, and P. Wielage, "A router architecture

for networks on silicon," *Proceedings of Progress*, vol. 2, 2001.

[17] L. Bononi, N. Concer, M. Grammatikakis, M. Coppola, and R. Locatelli, "Noc topologies exploration based on mapping and simulation models," in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on.* IEEE, 2007, pp. 543–546.

[18] M. M. Kim, J. D. Davis, M. Oskin, and T. Austin, "Polymorphic on-chip networks," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on.* IEEE, 2008, pp. 101–112.

[19] S. Tota, M. R. Casu, and L. Macchiarulo, "Implementation analysis of noc: a mpsoc trace-driven approach," in *Proceedings of the 16th ACM Great Lakes symposium on VLSI.* ACM, 2006, pp. 204–209.

[20] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture.* IEEE Computer Society, 2007, pp. 172–182.

[21] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Cost-efficient dragonfly topology for large-scale systems," in *Optical Fiber Communication Conference.* Optical Society of America, 2009, p. OTuI2.

[22] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proceedings of the conference on Design, automation and test in Europe.* ACM, 2000, pp. 250–256.

[23] A. S. Vaidya, A. Sivasubramaniam, and C. R. Das, "Impact of virtual channels and adaptive routing on application performance," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 2, pp. 223–237, 2001.

[24] H. Samuelsson and S. Kumar, "Ring road noc architecture," in *Norchip Conference, 2004. Proceedings.* IEEE, 2004, pp. 16–19.

[25] F. Karim, A. Nguyen, and S. Dey, "An interconnect architecture for networking systems on chips," *IEEE micro*, no. 5, pp. 36–45, 2002.

[26] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection networks: an engineering approach.* Morgan Kaufmann, 2003.

[27] V. Rantala, T. Lehtonen, and J. Plosila, *Network on chip routing algorithms.* Citeseer, 2006.

[28] J. Kim and H. Kim, "Router microarchitecture and scalability of ring topology in on-chip networks," in *Proceedings of the 2nd international workshop on network on chip architectures.* ACM, 2009, pp. 5–10.

[29] D. Siguenza-Tortosa and J. Nurmi, "Vhdl-based simulation environment for proteo noc," in *High-Level Design Validation and Test Workshop, 2002. Seventh IEEE International.* IEEE, 2002, pp. 1–6.

[30] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *IEEE transactions on Computers*, vol. 54, no. 8,

pp. 1025–1040, 2005.

[31] A. E. Kiasari, Z. Lu, and A. Jantsch, "An analytical latency model for networks-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 113–123, 2013.

[32] Y. M. Boura and C. R. Das, "Modeling virtual channel flow control in hypercubes," in *High-Performance Computer Architecture, 1995. Proceedings., First IEEE Symposium on*. IEEE, 1995, pp. 166–175.

[33] S. Foroutan, Y. Thonnart, R. Hersemeule, and A. Jerraya, "An analytical method for evaluating network-on-chip performance," in *proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 1629–1632.

[34] U. Y. Ogras and R. Marculescu, "Analytical router modeling for networks-on-chip performance analysis," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[35] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Cycle-accurate network on chip simulation with noxim," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 27, no. 1, p. 4, 2016.

[36] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture*

News, vol. 39, no. 2, pp. 1–7, 2011.

[37] V. S. Pai, P. Ranganathan, and S. V. Adve, "Rsim: An execution-driven simulator for ilp-based shared-memory multiprocessors and uniprocessors," in *Proceedings of the Third Workshop on Computer Architecture Education*, vol. 178. Citeseer, 1997.

[38] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins, "Interconnection networks enable fine-grain dynamic multi-tasking on fpgas," in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream.* Springer, 2002, pp. 795–805.

[39] T. Bartic, J.-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins, "Highly scalable network on chip for reconfigurable systems," in *System-on-Chip, 2003. Proceedings. International Symposium on.* IEEE, 2003, pp. 79–82.

[40] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri, "Lipar: A light-weight parallel router for fpga-based networks-on-chip," in *Proceedings of the 15th ACM Great Lakes symposium on VLSI.* ACM, 2005, pp. 452–457.

[41] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *INTEGRATION, the VLSI journal*, vol. 38, no. 1, pp. 69–93, 2004.

[42] L. Ost, A. Mello, J. Palma, F. Moraes, and N. Calazans, "Maia: a framework for networks on chip generation and verification," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*. ACM, 2005, pp. 49–52.

[43] C. A. Zeferino, M. E. Kreutz, and A. A. Susin, "Rasoc: A router soft-core for networks-on-chip," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 3. IEEE, 2004, pp. 198–203.

[44] C. Hilton and B. Nelson, "Pnoc: a flexible circuit-switched noc for fpga-based systems," in *Computers and Digital Techniques, IEE Proceedings-*, vol. 153, no. 3. IET, 2006, pp. 181–188.

[45] M. P. Véstias and H. C. Neto, "Area and performance optimization of a generic network-on-chip architecture," in *Proceedings of the 19th annual symposium on Integrated circuits and systems design*. ACM, 2006, pp. 68–73.

[46] K. Latif, A.-M. Rahmani, L. Guang, T. Seceleanu, and H. Tenhunen, "Pvs-noc: Partial virtual channel sharing noc architecture," in *2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*. IEEE, 2011, pp. 470–477.

[47] K. Latif, A.-M. Rahmani, E. Nigussie, T. Seceleanu, M. Radetzki, and H. Tenhunen, "Partial virtual channel sharing: a generic methodology to enhance resource management and fault tolerance in

networks-on-chip," *Journal of electronic testing*, vol. 29, no. 3, pp. 431–452, 2013.

[48] M. Shahiri, M. Valinataj, and A. M. Rahmani, "A reliable and high-performance network-on-chip router through decoupled resource sharing," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 88–95.

[49] R. Nau, "Notes on linear regression analysis," *Retrieved March*, vol. 5, p. 2016, 2014.

Vitae

* Name: Gamil Abdullah Mohsen Ahmed
* Nationality: Yemeni
* Date of Birth: 1982
* Email: *g201302310@kfupm.edu.sa , jameel.almoliki@gmail.com*
* Permenant Address: Ibb, Yemen